

IN THE  
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Robert A. Lester et al.

Confirmation No.:

Application No.: 09/967,155

Examiner: Vu, Trisha U.

Filing Date: 09/28/2001

Group Art Unit: 2112

Title: METHOD FOR IMPROVING PROCESSOR PERFORMANCE

Mail Stop Appeal Brief-Patents  
Commissioner For Patents  
PO Box 1450  
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Sir:

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on 01/04/2005.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) \$500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

( ) (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d) for the total number of months checked below:

( ) one month	\$120.00
( ) two months	\$450.00
( ) three months	\$1020.00
( ) four months	\$1590.00

( ) The extension fee has already been filled in this application.

(X) (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of \$500.00. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

(X) I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450. Date of Deposit: 03/11/2005  
OR

( ) I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number \_\_\_\_\_ on \_\_\_\_\_

Number of pages:

Typed Name: Robert A. Manware

Signature:

Respectfully submitted,

Robert A. Lester et al.

By

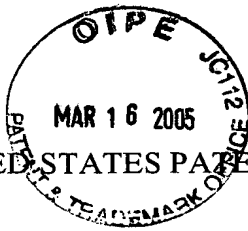
Robert A. Manware

Attorney/Agent for Applicant(s)

Reg. No. 48,758

Date: 03/11/2005

Telephone No.: (281) 970-4545



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Robert A. Lester et al.

Serial No.: 09/967,155

Filed: September 28, 2001

For: METHOD FOR IMPROVING  
PROCESSOR PERFORMANCE

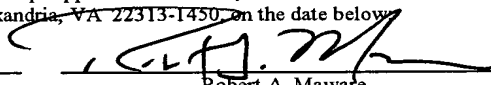
§  
§  
§  
§  
§  
§  
§  
§  
§

Group Art Unit: 2112

Examiner: Vu, Trisha U.

Atty. Docket: COMP:0233/FLE  
200302161-1

Mail Stop Appeal Brief-Patents  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

CERTIFICATE OF MAILING 37 C.F.R. 1.8	
I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date below:	
March 11, 2005	
Date	Robert A. Maware

**APPEAL BRIEF PURSUANT TO 37 C.F.R. §§ 41.31 AND 41.37**

This Appeal Brief is being filed in furtherance to the Notice of Appeal mailed on January 4, 2005, and received by the Patent Office on January 11, 2005.

The Commissioner is authorized to charge the requisite fee of \$500.00, and any additional fees which may be necessary to advance prosecution of the present application, to Account No. 08-2025, Order No. 200302161-1/FLE (COMP:0233).

03/17/2005 MAHMED1 00000039 082025 09967155

01 FC:1402 500.00 DA

1. **REAL PARTY IN INTEREST**

The real party in interest is Hewlett-Packard Development Company, L.P., a Texas Limited Partnership having its principal place of business in Houston, Texas and the Assignee of the above-referenced application. The Assignee of the above-referenced application will be directly affected by the Board's decision in the pending appeal.

2. **RELATED APPEALS AND INTERFERENCES**

Appellants are unaware of any other appeals or interferences related to this Appeal. The undersigned is Appellants' legal representative in this Appeal.

3. **STATUS OF CLAIMS**

Claims 1-13, 19-23 and 30-34 are currently pending, are currently under final rejection and, thus, are the subject of this appeal.

4. **STATUS OF AMENDMENTS**

As the instant claims have not been amended at any time, there are no outstanding amendments to be considered by the Board.

5. **SUMMARY OF CLAIMED SUBJECT MATTER**

The present invention relates generally to methods for improving processor performance. In accordance with embodiments of the present inventions, one technique for reducing request cycle time involves implementing an early "deferred reply" signal when a host controller is unable to process a request immediately. Page

11, lines 1-2. Advantageously, in accordance with embodiments of the present invention and in contrast to prior techniques, a more efficient method of processing requests is achieved by issuing a deferred reply to a requesting bus when the processor controller is free to process the request, regardless of whether the data has been retrieved from the memory and delivered to the host controller. Page 11, line 22 – page 12, line 1. Thus, once the requesting bus has enough bandwidth to handle the request, the host controller issues the deferred reply. Page 12, lines 1-3. A deferred reply may be issued as early as the clock cycle immediately subsequent to the clock cycle in which the request was originally deferred. Page 12, lines 3-4. Once the wait period has expired, the data may be waiting in the host controller for immediate delivery onto the requesting bus. Page 12, lines 9-12. By using the natural delays associated with standard protocols to carry out tasks (previously performed in series, as described above with reference to conventional techniques) in parallel, there may be a reduction in the latency associated with processing the request. Page 12, lines 12-14; *See* Fig. 3.

With regard to the aspect of the invention set forth in independent claim 1, discussions of the recited features of claim 1 can be found at least in the locations in the specification and drawings cited below. By way of example, an embodiment in accordance with the present invention relates to a method of processing a request in a computer system (e.g., 10). *See, e.g.*, page 22, line 1 – page 13, line 21; *see also* Fig. 3. The method comprises the act of initiating a read request from a requesting agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B), the requesting agent residing on a bus (e.g., 14A, 14B, 27), wherein the read request has an address corresponding to a memory location. *See, e.g.*, Fig. 3, block 40; *see also* page 12, lines 17-18. The

method also comprises the act of receiving the read request at a processor controller (e.g., PCON0, PCON1, PCON 2). *See, e.g.*, Fig. 3, block 42; *see also* page 12, lines 18-20. The method also comprises the act of sending the read request from the processor controller (e.g., PCON0, PCON1, PCON 2) to an access controller (e.g., MCON). *See, e.g.*, Fig. 3, block 50; *see also* page 13, lines 5-6. The method also comprises the act of sending a deferred reply from the processor controller (e.g., PCON0, PCON1, PCON 2) to the requesting agent (e.g., MCON) when the processor controller (e.g., PCON0, PCON1, PCON 2) is free to process the read request, wherein the deferred reply is sent before data corresponding to the read request is delivered to the access controller. *See, e.g.*, Fig. 3, block 56; *see also* page 13, lines 8-15. The method also comprises the act of delivering data residing at the address corresponding to the memory location to the access controller (e.g., MCON). *See, e.g.*, Fig. 3, block 52; *see also* page 13, line 6. The method also comprises the act of delivering the data from the access controller (e.g., MCON) to the processor controller (e.g., PCON0, PCON1, PCON 2). *See, e.g.*, Fig. 3, block 54; *see also* page 13, lines 6-8. The method also comprises the act of delivering the data from the processor controller (e.g., PCON0, PCON1, PCON 2) to the requesting agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B). *See, e.g.*, Fig. 3, block 60; *see also* page 13, lines 19-21.

With regard to the aspect of the invention set forth in independent claim 11, discussions of the recited features of claim 11 can be found at least in the locations in the specification and drawings cited below. By way of example, an embodiment in accordance with the present invention relates to a method of processing a request in a

computer system (e.g., 10). *See, e.g.*, page 22, line 1 – page 13, line 21; *see also* Fig.

3. The method comprises the act of sending a request from a processor controller (e.g., PCON0, PCON1, PCON 2) to an access controller (e.g., MCON) on a first clock cycle, the request originating from an agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B). *See, e.g.*, Fig. 3; *see also* page 12, lines 3-9. The method also comprises the act of sending a deferred reply from the processor controller (e.g., PCON0, PCON1, PCON 2) to the agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) on a second clock cycle, the second clock cycle being immediately subsequent to the first clock cycle). *See, e.g.*, Fig. 3; *see also* page 12, lines 3-9.

With regard to the aspect of the invention set forth in independent claim 19, discussions of the recited features of claim 19 can be found at least in the locations in the specification and drawings cited below. By way of example, an embodiment in accordance with the present invention relates to a computer system (e.g., 10). The computer system (e.g., 10) comprises a plurality of buses (e.g., 14A, 14B, 27). The computer system (e.g., 10) also comprises a memory system (e.g., 26) operably coupled to the plurality of buses (e.g., 14A, 14B, 27). The computer system (e.g., 10) also comprises a processor controller (e.g., PCON0, PCON1, PCON 2) coupled to each of the plurality of buses (e.g., 14A, 14B, 27) and configured to simultaneously issue a deferred reply to a requesting device (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) in response to receiving a read request from the requesting device (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) and obtain the data corresponding to the read request from the memory system (e.g., 26). *See, e.g.*, Fig. 3; *see also* page 12, lines 3-15; *see also* page 13, lines 12-15.

With regard to the aspect of the invention set forth in independent claim 30, discussions of the recited features of claim 30 can be found at least in the locations in the specification and drawings cited below. By way of example, an embodiment in accordance with the present invention relates to a computer system (e.g., 10). The computer system (e.g., 10) comprises a memory system (e.g., 26). The computer system (e.g., 10) also comprises a requesting agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) operably coupled to the memory system (e.g., 26) and configured to initiate read requests to the memory system (e.g., 26). The computer system (e.g., 10) also comprises a processor controller (e.g., PCON0, PCON1, PCON 2) coupled between the memory system (e.g., 26) and the requesting agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) and configured to send a deferred reply from the processor controller (e.g., PCON0, PCON1, PCON 2) to the requesting agent (e.g., 12A-12H, 32A-32B, 34A-34B, 36A-36B) when the processor controller (e.g., PCON0, PCON1, PCON 2) is free to process the read request, regardless of whether data corresponding to the read request has been delivered from the memory system (e.g., 26) to the processor controller (e.g., PCON0, PCON1, PCON 2). *See, e.g., Fig. 3; see also page 12, lines 3-15; see also page 13, lines 12-15.*

In summary, by issuing the deferred reply to the requesting bus when the processor controller is free to process the request, regardless of whether the data has been retrieved from the memory and delivered to the host controller, a more efficient method of processing requests may be achieved. Page 11, line 22 – page 12, line 1. Once the requesting bus has enough bandwidth to handle the request, the host controller

issues the deferred reply. Page 12, lines 1-3. A deferred reply may be issued as early as the clock cycle immediately subsequent to the clock cycle in which the request was originally deferred. Page 12, lines 3-4. By initiating the deferred reply immediately rather than waiting for the data to be fetched, the latent clock cycles associated with the issuance of the deferred reply and defined by the system protocol can be used in transferring the data from memory to the host controller. Page 12, lines 6-9.

Accordingly, once the wait period has expired, the data may be waiting in the host controller for immediate delivery onto the requesting bus. Page 12, lines 9-12. By using the natural delays associated with standard protocols to carry out tasks (previously performed in series, as described above with reference to conventional techniques) in parallel, there may be a reduction in the latency associated with processing the request. Page 12, lines 12-14; *See Fig. 3.*

6. **GROUND OF REJECTION TO BE REVIEWED ON APPEAL**

**First Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's first ground of rejection in which the Examiner rejected claims 1, 3-7, 9-10, 19-21, 30, 32 and 33 under 35 U.S.C. § 102(e) as being anticipated by Hunsaker (U.S. Pub No. 2003/0037198).

**Second Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's second ground of rejection in which the Examiner rejected claims 2 and 11-13 under

35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) in view of Solomon (U.S. Pat. No. 6,647,454).

**Third Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's third ground of rejection in which the Examiner rejected claim 8 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Ajanovic et al. (U.S. Patent No. 5,761,444).

**Fourth Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's third ground of rejection in which the Examiner rejected claim 22 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Jayakumar et al. (U.S. Patent No. 6,012,118).

**Fifth Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's third ground of rejection in which the Examiner rejected claim 23 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Stallmo et al. (U.S. Patent No. 5,613,059).

**Sixth Ground of Rejection for Review on Appeal:**

Appellants respectfully urge the Board to review and reverse the Examiner's third ground of rejection in which the Examiner rejected claim 31 under 35 U.S.C. §

103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Thekkath et al. (U.S. Patent No. 6,681,283).

7. **ARGUMENT**

As discussed in detail below, the Examiner has improperly rejected the pending claims. Further, the Examiner has misapplied long-standing and binding legal precedents and principles in rejecting the claims under Sections 102 and 103. Accordingly, Appellants respectfully request full and favorable consideration by the Board, as Appellants strongly believe that claims 1-34 are currently in condition for allowance.

A. **Ground of Rejection No. 1:**

The Examiner rejected claims 1, 3-7, 9-10, 19-21, 30, 32 and 33 under 35 U.S.C. § 102(e) as being anticipated by Hunsaker (U.S. Pub No. 2003/0037198). Each of the independent claims will be discussed separately below. Appellants respectfully traverse this rejection.

1. **Judicial precedent has clearly established a legal standard for a prima facie anticipation rejection.**

Anticipation under Section 102 can be found only if a single reference shows exactly what is claimed. *Titanium Metals Corp. v. Banner*, 227 U.S.P.Q. 773 (Fed. Cir. 1985). Thus, for a prior art reference to anticipate under Section 102, every element of the claimed invention must be identically shown in a single reference. *In re Bond*, 15 U.S.P.Q.2d 1566 (Fed. Cir. 1990). Moreover, the prior art reference also must show the *identical* invention “*in as complete detail as contained in the ... claim*”

to support a *prima facie* case of anticipation. *Richardson v. Suzuki Motor Co.*, 9 U.S.P.Q. 2d 1913, 1920 (Fed. Cir. 1989) (emphasis added). Accordingly, Appellants need only point to a single element not found in the cited reference to demonstrate that the cited reference fails to anticipate the claimed subject matter.

2. **The Examiner's rejection of independent claim 1 is improper because the rejection fails to establish a prima facie case of anticipation.**

Independent claim 1 recites:

A method of processing a request in a computer system, comprising the acts of:

(a) initiating a read request from a requesting agent, the requesting agent residing on a bus, wherein the read request has an address corresponding to a memory location;

(b) receiving the read request at a processor controller;

(c) sending the read request from the processor controller to an access controller;

(d) sending a deferred reply from the processor controller to the requesting agent when the processor controller is free to process the read request, wherein the deferred reply is sent before data corresponding to the read request is delivered to the access controller;

(e) delivering data residing at the address corresponding to the memory location to the access controller;

(f) delivering the data from the access controller to the processor controller; and

(g) delivering the data from the processor controller to the requesting agent.

In rejecting independent claim 1, the Examiner asserted that the Hunsaker reference discloses all of the recited features of the claim. *See* Official Action mailed October 10, 2004, pages 2-3. Appellants respectfully traverse this assertion.

As discussed in the present application, in certain instances, a host controller (which includes one or more processor controllers PCON and a memory controller MCON) is unable to immediately process a request received from a requesting agent. Page 10, line 13; Fig. 2. In this case, rather than returning the requested data, the host controller defers the cycle, freeing it from the bus and indicating that the cycle will be completed at a later time. Page 10, lines 15-17. As discussed in the present application, in accordance with conventional techniques, if a request is received at a host controller and the request is unable to be processed immediately, the host controller defers the cycle to the requesting agent and retrieves the data from memory. Page 10, lines 15-17. Once the data is retrieved from the memory, it is temporarily stored in the host controller, and the host controller issues a “deferred reply” to the requesting bus. *See* page 10, lines 17-19 and page 11, lines 13-15.

As will be appreciated by those skilled in the art, a “deferred reply” is generally issued after the data is retrieved from memory and when the host controller and requesting bus are ready to complete the transaction. Once the bus is able to handle the data and deliver it to the requesting agent, the requested data is delivered to the requesting agent. Page 11, lines 14-15. Because of processor specific timing requirements and the associated architectural protocol, once a deferred reply is issued, a host controller waits some period of time after the issuance of the deferred reply until the appropriate data can be sent from the host controller to the requesting agent. Page 10, lines 19-23. Accordingly, by issuing the deferred reply only after data has been retrieved from memory and delivered to the host controller, cycle time may be

added to the processing of the request due to the processor specific timing requirements and the associated architectural protocol.

As will be appreciated by those skilled in the art, the Hunsaker reference simply describes the conventional techniques discussed above. Specifically, the Hunsaker reference discloses receiving a read request at a PCI bridge and issuing a delayed transaction or receiving a read request at a PCI-X bridge and issuing a split transaction. Paragraph [0004], lines 1-5. When a PCI bridge receives the read data from another device, it is stored in a transaction buffer in the PCI bridge. Paragraph [0004], lines 8-9. Once the same read request is **resent** by the requesting agent and received at the PCI bridge, the data is delivered to the requesting agent. Paragraph [0004], lines 8-10. Similarly, a PCI-X split transaction occurs when an I/O device issues an initial read request. Paragraph [0004], lines 9-11. Since the PCI-X bridge does not have the read data, it terminates the transaction with a split response indicating that the bridge has accepted the read request and will later provide the I/O device with the read completion data. Paragraph [0004], lines 11-15. When the bridge receives the read completion data, it stores it in a delayed transaction buffer. Paragraph [0004], lines 15-17. The bridge *then* sends the data to the I/O device as a split completion transaction. Paragraph [0004], lines 17-18.

In contrast to the conventional prior techniques described in the present application and disclosed in the Hunsaker reference, in accordance with an exemplary embodiment of the present invention, a more efficient method of processing requests by issuing the deferred reply to the requesting bus when the processor controller is free

to process the request, regardless of whether the data has been retrieved from the memory and delivered to the host controller. Page 11, line 22 – page 12, line 1. Thus, once the requesting bus has enough bandwidth to handle the request, the host controller issues the deferred reply. Page 12, lines 1-3. A deferred reply may be issued as early as the clock cycle immediately subsequent to the clock cycle in which the request was originally deferred. Page 12, lines 3-4. By initiating the deferred reply upon the availability of the processor controller to process the request, the latent clock cycles associated with the issuance of the deferred reply and defined by the system protocol can be used in transferring the data from memory to the host controller (e.g., to the memory controller MCON). Page 12, lines 6-9. Accordingly, once the wait period has expired, the data may be waiting in the host controller for delivery onto the requesting bus. Page 12, lines 9-12. By using the natural delays associated with standard protocols to carry out tasks (previously performed in series, as described above with reference to conventional techniques) in parallel, there may be a reduction in the latency associated with processing the request. Page 12, lines 12-14; *See Fig. 3.*

The Examiner appears to be asserting that the split transaction of the PCI-X bus described in the Hunsaker reference indicates the same parallel processing described above. In other words, it appears that the Examiner believes that the conventional split transaction associated with PCI-X architectures provides for issuing a deferred reply from the host controller to the requesting agent, before the corresponding data is delivered from the memory to the host controller, as in claim 1. Appellants respectfully traverse this understanding.

As will be appreciated by those skilled in the art, the “Split Transaction” associated with the PCI-X architecture includes a “Split Response” and a “Split Completion.” *See e.g.,* Hunsaker, paragraph [0004], lines 11-19. The description in Hunsaker is consistent with the common meanings of the terms understood by those skilled in the art and described throughout the PCI-X specification, portions of which are attached hereto as “Exhibit A.” Specifically, pages 35 and 36 of the PCI-X specification provide definitions of the terms associated with the Split Transaction. Further, pages 127-140 of the PCI-X specification describe Split Transactions. Pages 141-157 of the PCI-X specification describe Transaction Termination. The Examiner appears to be asserting that the “Split Transaction” and more specifically, the “Split Response” is correlative to the recited “Deferred Reply.” In the context of computer systems, “response” and “reply” are not synonymous, as exemplified below with reference to PCI and PCI-X protocols. The following discussion is fully supported by the PCI and PCI-X specifications, which are hereby incorporated by reference.

Contrary to the Examiner’s assertions, at best, the Split Completion portion of the disclosed Split Transaction might hypothetically be considered analogous to the presently recited “deferred reply.” That is, the “Split Completion” transaction of Hunsaker is delivered from the PCI-X bridge to the requesting agent to indicate that the PCI-X bridge is now ready to complete the transaction. As with the deferred reply, the “Split Completion” has a certain inherent delay associated therewith. As such, once the Split Completion is issued by the PCI-X bridge, the bridge waits a

number of clock cycles (associated with processor specific timing requirements and the associated architectural protocol) before actually transferring the data from the PCI-X bridge to the requesting agent. As described in paragraph [0004] of the Hunsaker reference, the Split Completion transaction is issued only after the data is retrieved from the memory and temporarily stored in the PCI-X bridge. Accordingly, the conventional techniques described in the Hunsaker reference are subject to the same delays associated with the serial methods discussed above, wherein the deferred reply/ Split Completion transaction is issued only **after** data has been retrieved from memory and delivered to the host controller/ PCI-X bridge.

By way of background, the present claims can be distinguished from PCI and PCI-X protocols. As will be appreciated by those skilled in the art, in accordance with the protocol used in the prior art, for transactions carried out over a processor (PCI) bus, a transaction comprises several phases, including an Arbitration phase, a Request phase, a Response phase, and a Data phase. When the initiator (i.e., requesting device or agent) issues a Read transaction, it must first arbitrate for the bus and then signal a Read command in the Request phase. If the target cannot furnish the read data immediately, but does not wish to tie up the bus, it may signal a Deferred response in the Response phase, indicating that it will return data in a subsequent transaction. At this point the Read transaction has been deferred and no longer ties up the bus. Later, when the target wants to return data, it must issue a Deferred Reply transaction. Again, it must arbitrate for the bus and then signal a Deferred Reply command in the Request phase, followed by a Normal response in the Response phase and the read data in the Data phase.

For transactions carried out over a PCI-X bus, a transaction also comprises several phases, including an Arbitration phase, an Address/Command phase, and a Data phase. When an initiator issues a Read transaction, it must first arbitrate for the bus and then signal a Read command in the Address/Command phase. If the target (completer) cannot furnish the read data immediately, but does not wish to tie up the bus, it may signal a Split Response to terminate the transaction, indicating that it will return data in a subsequent transaction. At this point the Read transaction has been split into two transactions - a Split Request transaction, which has just been terminated with a Split Response, followed later by a Split Completion transaction when the target wants to return data. Again, the target must arbitrate for the bus and then signal a Split Completion command in the Address/Command phase, followed by the read data in the Data phase with a normal termination.

It should be noted that on both of these buses, the Read request involves two bus transactions. On the processor bus, the Read transaction gets a Deferred Response, followed by a Deferred Reply transaction which returns the data. On the PCI-X bus, the Read transaction gets a Split Response, followed by a Split Completion transaction which returns the data.

Embodiments of the present invention, as recited in the present claims, facilitate an improvement which occurs between when the target signals a Deferred Response in the Response phase of the original Read transaction, and when the target issues the second Deferred Reply transaction. In accordance with the improved

techniques, the processor controller does not wait until the read data is available before starting the Deferred Reply transaction by arbitrating for the bus and signaling a Deferred Reply command in the Request phase. Accordingly, in comparison to exemplary PCI and PCI-X protocols, by the time the read data is returned to the processor controller, the Read transaction is already in the Data phase, so the read data can be more rapidly returned to the initiator (requesting agent) along with a Normal Response.

Again, as indicated in the Split Transaction description on pages 127-128 in the PCI-X specification, typically, the target (completer) waits for the read data before issuing the second Split Completion transaction:

After signaling Split Response, the completer executes the transaction...If the transaction is a read, the completer prepares all or some of the bytes specified by the byte count (for burst reads) or byte enables (for DWORD reads) of the Split Request. The completer initiates a Split Completion transaction to send the requested read data or a completion message to the requester

Accordingly, Appellants respectfully submit that the Hunsaker reference does not disclose the features recited in independent claims 1. Specifically, Appellants respectfully submit that the Hunsaker reference does not disclose “sending a deferred reply from the processor controller to the requesting agent when the processor controller is free to process the read request, *wherein the deferred reply is sent before data corresponding to the read request is delivered to the access controller,*” as recited in claim 1. Emphasis added. As described at length above, the recited “deferred reply” is simply *not* correlative to aspects of the Split Transaction, such as the Split Response. The Hunsaker reference, nor any other art of record does not

disclose sending a deferred reply, “wherein the deferred reply is sent before data corresponding to the read request is delivered to the access controller.” Accordingly, the Hunsaker reference cannot possibly anticipate the recited subject matter of claim 1.

Because the reference fails to disclose each element recited by the instant claim, the Hunsaker reference fails to anticipate independent claim 1. Thus, the Examiner’s rejection of independent claim 1, and claims 3-7, 9 and 10, depending therefrom, is clearly improper. Accordingly, Appellants request the Board overturn the rejection and allow independent claim 1 as well as dependent claims 3-7, 9 and 10.

3. **The Examiner’s rejection of independent claim 19 is improper because the rejection fails to establish a prima facie case of anticipation.**

Independent claim 19 recites:

A computer system comprising:  
a plurality of buses;  
a memory system operably coupled to the plurality of buses; and  
a processor controller coupled to each of the plurality of buses and configured to simultaneously issue a deferred reply to a requesting device in response to receiving a read request from the requesting device and obtain the data corresponding to the read request from the memory system.

The Hunsaker reference fails to disclose every element of independent claim 19. Appellants note that independent claim 19 recites “a processor controller coupled to each of the plurality of buses and configured to *simultaneously issue a deferred reply* to a requesting device in response to receiving a read request from the

requesting device *and obtain the data* corresponding to the read request from the memory system.” Emphasis added. As summarized above with respect to the improper rejection of claim 1, the arguments for which are incorporated herein by reference, Hunsaker does not disclose simultaneously issuing a deferred reply and obtaining data.

Because the reference fails to disclose each element recited by the instant claim, the Hunsaker reference fails to anticipate independent claim 19. Thus, the Examiner’s rejection of independent claim 19, and claims 20 and 21 depending therefrom, is clearly improper. Accordingly, Appellants request the Board overturn the rejection and allow independent claim 19 as well as dependent claims 20 and 21.

4. **The Examiner’s rejection of independent claim 30 is improper because the rejection fails to establish a prima facie case of anticipation.**

Independent claim 30 recites:

A computer system comprising:  
a memory system;  
a requesting agent operably coupled to the memory system and configured to initiate read requests to the memory system; and  
a processor controller coupled between the memory system and the requesting agent and configured to send a deferred reply from the processor controller to the requesting agent when the processor controller is free to process the read request, regardless of whether data corresponding to the read request has been delivered from the memory system to the processor controller.

The Hunsaker reference fails to disclose every element of independent claim 30. Appellants note that independent claim 30 recites “a processor controller coupled between the memory system and the requesting agent and *configured to send a deferred reply* from the processor controller to the requesting agent when the

processor controller is free to process the read request, *regardless of whether data corresponding to the read request has been delivered from the memory system to the processor controller.*” Emphasis added. As summarized above with respect to the improper rejection of claim 30, the arguments for which are incorporated herein by reference, Hunsaker does not disclose sending a deferred reply, regardless of whether data corresponding to the read request has been delivered from the memory system to the processor controller.

Because the reference fails to disclose each element recited by the instant claim, the Hunsaker reference fails to anticipate independent claim 30. Thus, the Examiner’s rejection of independent claim 30, and claims 32 and 33 depending therefrom, is clearly improper. Accordingly, Appellants request the Board overturn the rejection and allow independent claim 30 as well as dependent claims 32 and 33.

B. **Ground of Rejection No. 2:**

1. **Judicial precedent has clearly established a legal standard for a prima facie obviousness rejection.**

The burden of establishing a *prima facie* case of obviousness falls on the Examiner. *Ex parte Wolters and Kuypers*, 214 U.S.P.Q. 735 (B.P.A.I. 1979). Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention absent some teaching or suggestion supporting the combination. *ACS Hospital Systems, Inc. v. Montefiore Hospital*, 732 F.2d 1572, 1577, 221 U.S.P.Q. 929, 933 (Fed. Cir. 1984). Accordingly, to establish a *prima facie* case, the Examiner must not only show that the combination includes all of the

claimed elements, but also a convincing line of reason as to why one of ordinary skill in the art would have found the claimed invention to have been obvious in light of the teachings of the references. *Ex parte Clapp*, 227 U.S.P.Q. 972 (B.P.A.I. 1985).

When prior art references require a selected combination to render obvious a subsequent invention, there must be some reason for the combination other than the hindsight gained from the invention itself, i.e., something in the prior art as a whole must suggest the desirability, and thus the obviousness, of making the combination. *Uniroyal Inc. v. Rudkin-Wiley Corp.*, 837 F.2d 1044, 5 U.S.P.Q.2d 1434 (Fed. Cir. 1988).

2. **The Examiner's rejection of independent claim 11 is improper because the rejection fails to establish a prima facie case of obviousness.**

The Examiner rejected claims 2 and 11-13 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198). Appellants respectfully traverse this rejection.

Independent claim 11 recites “sending a request from a processor controller to an access controller on a first clock cycle,” and “sending a deferred reply from the processor controller to the agent on a second clock cycle, the second clock cycle being immediately subsequent to the first clock cycle.” As discussed at length above, the Hunsaker reference does not disclose receiving a request at a host controller and immediately sending a deferred reply. As discussed above with regard to the rejections under 35 U.S.C. § 102, the Hunsaker reference discloses waiting for the data to be retrieved from the memory before issuing a split completion transaction

(deferred reply). Contrary to the Examiner's assertion, the Solomon reference does not cure this deficiency. As with the Hunsaker reference, the Solomon reference merely discusses inherent aspects of a split transaction, as related to a PCI bus. The Examiner appears to assert that the "split response termination" transaction is the same as the presently recited "deferred reply." As will be appreciated by those skilled in the art, this correlation is not accurate. The techniques described in the Solomon reference will still be subject to the delays described above. That is, once the data is retrieved from memory, a deferred reply or split completion will be sent to the requesting agent. Thus, the Solomon reference does not disclose sending a deferred reply immediately after requesting the data from memory.

Because neither of the references alone or in combination discloses each of the features recited in independent claim 11, the cited combination cannot possibly render the recited subject matter obvious. Accordingly, Appellants respectfully request that the Board withdraw the obviousness rejections and allow claims 11 and 12 and 13, depending therefrom. Further, Appellants respectfully request that the Board withdraw the obviousness rejection and allow claim 2, which is dependent on claim 1.

**C. Ground of Rejection No. 3:**

The Examiner rejected claim 8 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Ajanovic et al. (U.S. Patent No. 5,761,444). Appellants respectfully traverse this rejection.

Claim 8 depends from one of independent claim 1 discussed above.

Moreover, each of the Examiner's obviousness rejections is based primarily on the Hunsaker reference, which is also discussed above. With this in mind, Appellants respectfully assert that the Ajanovic et al. reference, employed in conjunction with the Hunsaker reference, does not obviate the deficiencies of the Hunsaker reference as discussed in the foregoing remarks regarding the Examiner's rejections of independent claim 1. Accordingly, Appellants respectfully assert that the instant claim is not only patentable for its dependencies on an allowable base claim but also by virtue of the additional features recited therein.

In light of the forgoing remarks, Appellants respectfully request that the Board withdraw the obviousness rejection in relation to claim 8. Additionally, Appellants respectfully request that the Board direct the Examiner to allow the instant claim.

**D. Ground of Rejection No. 4:**

The Examiner rejected claim 22 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Jayakumar et al. (U.S. Patent No. 6,012,118). Appellants respectfully traverse this rejection.

Claim 22 depends from one of independent claim 19 discussed above.

Moreover, each of the Examiner's obviousness rejections is based primarily on the Hunsaker reference, which is also discussed above. With this in mind, Appellants respectfully assert that the Jayakumar et al. reference, employed in conjunction with

the Hunsaker reference, does not obviate the deficiencies of the Hunsaker reference as discussed in the foregoing remarks regarding the Examiner's rejections of independent claim 19. Accordingly, Appellants respectfully assert that the instant claim is not only patentable for its dependencies on an allowable base claim but also by virtue of the additional features recited therein.

In light of the foregoing remarks, Appellants respectfully request that the Board withdraw the obviousness rejection in relation to claim 22. Additionally, Appellants respectfully request that the Board direct the Examiner to allow the instant claim.

E. **Ground of Rejection No. 5:**

The Examiner rejected claim 23 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Stallmo et al. (U.S. Patent No. 5,613,059). Appellants respectfully traverse this rejection.

Claim 23 depends from one of independent claim 19 discussed above. Moreover, each of the Examiner's obviousness rejections is based primarily on the Hunsaker reference, which is also discussed above. With this in mind, Appellants respectfully assert that the Stallmo et al. reference, employed in conjunction with the Hunsaker reference, does not obviate the deficiencies of the Hunsaker reference as discussed in the foregoing remarks regarding the Examiner's rejections of independent claim 19. Accordingly, Appellants respectfully assert that the instant

claim is not only patentable for its dependencies on an allowable base claim but also by virtue of the additional features recited therein.

In light of the forgoing remarks, Appellants respectfully request that the Board withdraw the obviousness rejection in relation to claim 23. Additionally, Appellants respectfully request that the Board direct the Examiner to allow the instant claim.

F. **Ground of Rejection No. 6:**

The Examiner rejected claim 31 under 35 U.S.C. § 103(a) as being unpatentable over Hunsaker (U.S. Pub No. 2003/0037198) and further in view of Thekkath et al. (U.S. Patent No. 6,681,283). Appellants respectfully traverse this rejection.

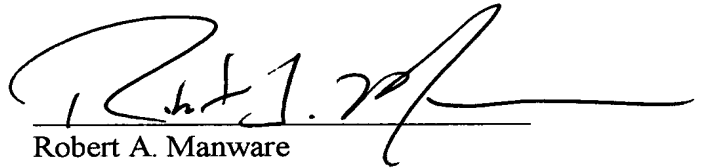
Claim 31 depends from one of independent claim 30 discussed above. Moreover, each of the Examiner's obviousness rejections is based primarily on the Hunsaker reference, which is also discussed above. With this in mind, Appellants respectfully assert that the Thekkath et al. reference, employed in conjunction with the Hunsaker reference, does not obviate the deficiencies of the Hunsaker reference as discussed in the foregoing remarks regarding the Examiner's rejections of independent claim 30. Accordingly, Appellants respectfully assert that the instant claim is not only patentable for its dependencies on an allowable base claim but also by virtue of the additional features recited therein.

In light of the forgoing remarks, Appellants respectfully request that the Board withdraw the obviousness rejection in relation to claim 31. Additionally, Appellants respectfully request that the Board direct the Examiner to allow the instant claim.

**Conclusion**

Appellants respectfully submit that all pending claims are in condition for allowance. However, if the Examiner or Board wishes to resolve any other issues by way of a telephone conference, the Examiner or Board is kindly invited to contact the undersigned attorney at the telephone number indicated below.

Respectfully submitted,

A handwritten signature in black ink, appearing to read 'R. A. Manware', written over a horizontal line.

Robert A. Manware  
Reg. No. 48,758  
(281) 970-4545

Date: March 11, 2005

**CORRESPONDENCE ADDRESS**  
HEWLETT-PACKARD COMPANY  
Intellectual Property Administration  
P.O. Box 272400  
Fort Collins, Colorado 80527-2400

8. **APPENDIX OF CLAIMS ON APPEAL**

**Listing of Claims:**

1. A method of processing a request in a computer system, comprising the acts of:

(a) initiating a read request from a requesting agent, the requesting agent residing on a bus, wherein the read request has an address corresponding to a memory location;

(b) receiving the read request at a processor controller;

(c) sending the read request from the processor controller to an access controller;

(d) sending a deferred reply from the processor controller to the requesting agent when the processor controller is free to process the read request, wherein the deferred reply is sent before data corresponding to the read request is delivered to the access controller;

(e) delivering data residing at the address corresponding to the memory location to the access controller;

(f) delivering the data from the access controller to the processor controller; and

(g) delivering the data from the processor controller to the requesting agent.

2. The method of processing a request, as set forth in claim 1, comprising the act of sending a data ready signal from the access controller to the processor controller upon receipt of the data at the access controller.

3. The method of processing a request, as set forth in claim 1, wherein act (a) comprises the act of initiating a read request from a processor residing on a processor bus.

4. The method of processing a request, as set forth in claim 1, wherein act (a) comprises the act of initiating a read request from a peripheral device residing on an input/output (I/O) bus.

5. The method of processing a request, as set forth in claim 1, wherein act (c) comprises the act of sending the request from the processor controller to a memory controller.

6. The method of processing a request, as set forth in claim 1, wherein act (c) comprises the act of sending the request from the processor controller to a tag controller.

7. The method of processing a request, as set forth in claim 1, wherein act (d) comprises the act of sending a deferred reply immediately upon the agent bus being available to receive data from the memory location.

8. The method of processing a request, as set forth in claim 1, wherein act (d) comprises the act of waiting a predetermined number of clock cycles after the deferred reply is sent before delivering data.

9. The method of processing a request, as set forth in claim 1, wherein acts (d) and (f) are performed simultaneously.

10. The method of processing a request, as set forth in claim 1, wherein the acts are performed in the recited order.

11. A method of processing a request in a computer system, comprising the acts of:

(a) sending a request from a processor controller to an access controller on a first clock cycle, the request originating from an agent; and

(b) sending a deferred reply from the processor controller to the agent on a second clock cycle, the second clock cycle being immediately subsequent to the first clock cycle.

12. The method of processing a request, as recited in claim 11, wherein act (a) comprises the act of sending a request from a processor controller to a memory controller on a first clock cycle.

13. The method of processing a request, as set forth in claim 11, wherein act (a) comprises the act of sending a request for a processor controller to a tag controller on a first clock cycle.

14 – 18. (Canceled)

19. A computer system comprising:  
  
a plurality of buses;  
  
a memory system operably coupled to the plurality of buses; and  
  
a processor controller coupled to each of the plurality of buses and configured to simultaneously issue a deferred reply to a requesting device in response to receiving a read request from the requesting device and obtain the data corresponding to the read request from the memory system.

20. The computer system, as set forth in claim 19, wherein at least one of the plurality of buses comprises a processor bus.

21. The computer system, as set forth in claim 19, wherein at least one of the plurality of buses comprises an input/output (I/O) bus.

22. The computer system, as set forth in claim 19, wherein the processor controller is further configured to obtain the data corresponding to the read request from the memory system before issuing a deferred reply to a requesting device.

23. The computer system, as set forth in claim 19, wherein the memory system comprises a redundant memory system.

24-29. (Canceled)

30. A computer system comprising:

a memory system;

a requesting agent operably coupled to the memory system and configured to initiate read requests to the memory system; and

a processor controller coupled between the memory system and the requesting agent and configured to send a deferred reply from the processor controller to the requesting agent when the processor controller is free to process the read request, regardless of whether data corresponding to the read request has been delivered from the memory system to the processor controller.

31. The computer system, as set forth in claim 30, wherein the requesting agent comprises a processor.

32. The computer system, as set forth in claim 30, wherein the requesting agent comprises an input/output (I/O) device.

33. The computer system, as set forth in claim 30, wherein the processor controller is configured to obtain the data corresponding to the read request from the memory system before issuing a deferred reply to the requesting agent.

34. The computer system, as set forth in claim 30, wherein the memory system comprises a redundant memory system.

**EXHIBIT**

**A**

<b>source bridge</b>	The bridge that creates a bus segment capable of PCI-X operation in a system hierarchy. The source bridge is required to initiate Type 0 configuration transactions on that bus segment. (Other devices optionally initiate Type 0 configuration transactions.) A host bridge is the source bridge for the PCI bus it creates. A PCI-X bridge is the source bridge for its secondary bus.
<b>source synchronous</b>	A data transfer method in which the device that is the source of the data also drives strobes to be used for latching the data at the destination. Source-synchronous transactions use source-synchronous sampling for the AD and ECC buses in PCI-X Mode 2 for data phases of burst push transactions other than Memory Write. They use common-clock sampling for the AD and ECC buses during all other data phases and all address and attribute phases. They use common-clock sampling for the control signal during all phases. See also "common clock."
<b>Split Completion</b>	<p>When used in the context of the bus protocol, this term refers to a transaction using the Split Completion command. It is used by the completer to send the requested data (for read transactions completed without error) or a completion message back to the requester.</p> <p>When used in the context of transaction ordering and the transaction queues inside the requester, completer, and bridges, the term refers to a queue entry corresponding to a Split Completion transaction on the bus.</p>
<b>Split Completion address</b>	Information driven by the completer or a bridge on the AD bus during the address phase of a Split Completion transaction. The Split Completion address includes the Requester ID and is used to route the Split Completion to the requester.
<b>Split Completion Message</b>	<p>In the context of bus transactions, a Split Completion Message is a Split Completion transaction that notifies the requester that a request (either read or write) encountered an error, or that a write request completed without an error.</p> <p>In the context of the Split Completion address, the term refers to the attribute bit that indicates the Split Completion is a message rather than data for a read request.</p>
<b>Split Request</b>	When used in the context of the bus protocol, this term refers to a transaction terminated with Split Response. When used in the context of transaction ordering and the transaction queues inside the requester, completer, and bridges, the term refers to a queue entry corresponding to a Split Request transaction on the bus. When the completer executes the Split Request, it becomes a Split Completion.

<b>Split Response</b>	Protocol for terminating a transaction, whereby the target indicates that it will complete the transaction as a Split Transaction. The target may optionally terminate any DWORD transaction (except Special Cycle) and any read transaction with Split Response.
<b>Split Transaction</b>	A single logical transfer containing an initial transaction (the Split Request) that the target (the completer or a bridge) terminates with Split Response, followed by one or more transactions (the Split Completions) initiated by the completer (or bridge) to send the read data (if a read) or a completion message back to the requester.
<b>starting address</b>	Address indicated in the address phase of all transactions except Split Completions, Interrupt Acknowledges, Special Cycles, and Device ID Messages. This is the first byte affected by the transaction. The starting address of all transactions except configuration transactions is permitted to be aligned to any byte (i.e., it uses the full address bus). The starting address of configuration transactions must be aligned to a DWORD boundary (i.e., AD[1::0] must be 0).  Split Completion transactions use only a partial starting address as described in Section 2.10.3. As in conventional PCI, Interrupt Acknowledge and Special Cycle transactions have no address.
<b>system</b>	Any functional combination of a system board, add-in cards, and/or software.
<b>system board</b>	A collection of hardware that includes a CPU, host bridge, and a PCI bus. In some system boards, the bus includes one or more devices in addition to the source bridge. In some system boards the bus connects to one or more add-in card slots.
<b>Tag</b>	A 5-bit number assigned by the initiator of a Sequence to distinguish it from other Sequences. It appears in the attribute field and is part of the Sequence ID.  An initiator must not reuse a Tag for a new Sequence until the original Sequence is complete (i.e., byte count satisfied, error condition encountered, etc.). (See Section 2.1 for more details.)
<b>target</b>	A device that responds to bus transactions. A bridge forwarding a transaction is the target on the originating bus.
<b>target data phase signaling</b>	The target signals one of the following on each data-phase clock: Split Response Target-Abort Single Data Phase Disconnect Wait State Data Transfer Retry Disconnect at Next ADB

## 2.10. Split Transactions

Split Transactions improve bus efficiency for transactions accessing targets that exhibit long latencies. Split Transactions in PCI-X systems replace Delayed Transactions in conventional PCI systems. Unlike conventional PCI, the target must not assume the initiator will repeat a transaction terminated with Retry. (In some cases, the device is obligated to continue the transaction, but the target must not depend upon the transaction being repeated verbatim. For example, if a completer encountered an error after a Split Completion transaction for a burst read that was terminated with Retry, the completer would be permitted to continue the Sequence with a Split Completion Error rather than repeating the original Split Completion.)

### 2.10.1. Basic Split Transaction Requirements

A Split Transaction consists of at least two separate bus transactions, a Split Request initiated by the requester, and one or more Split Completions initiated by the completer.

Transactions using any of the following commands are permitted to use Split Transactions:

- ☐ Memory Read Block
- ☐ Alias to Memory Read Block
- ☐ Memory Read DWORD
- ☐ Interrupt Acknowledge
- ☐ I/O Read
- ☐ I/O Write
- ☐ Configuration Read
- ☐ Configuration Write

The target of such a transaction may optionally complete the transaction as a Split Transaction or may use any other termination method as determined by the rules for those termination methods. All of these termination alternatives are available regardless of whether the transaction was previously terminated with Retry. (See Section 2.11.2.5 for examples of implementations in which the device is unable to respond with Split Response and signals Target-Abort.) Once the target terminates a read transaction with Split Response, the target must transfer the entire requested byte count as a Split Completion (except for error conditions described in Section 2.10.6.2 and Section 8.8).

A Split Transaction begins when the requester initiates a transaction using one of the commands in the list above. The completer optionally signals Split Response as defined in Section 2.11.2.4. (PCI-X bridges are required to signal Split Response in some cases. See Section 8.4.)

Target initial wait states for Split Response termination are allowed up to the limit specified in Section 2.9.1. A transaction terminated with Split Response is called a Split Request.

After signaling Split Response, the completer executes the transaction. If the transaction is a write, the completer updates the bytes specified by the byte enables of the Split Request. If

the transaction is a read, the completer prepares all or some of the bytes specified by the byte count (for burst reads) or byte enables (for DWORD reads) of the Split Request. The completer initiates a Split Completion transaction to send the requested read data or a completion message to the requester. Notice that for a Split Completion transaction, the requester and the completer switch roles. The completer becomes the initiator of the Split Completion transaction, and the requester becomes the target.

A device's ability to execute a transaction as a Split Transaction is unaffected by the state of the Bus Master bit in the Command register. A device that is properly addressed by a transaction is permitted to terminate that transaction with Split Response, request the bus, and initiate a Split Completion even if its Bus Master bit in the Command register is cleared.

A Split Transaction is not finished until the requester receives Split Completion transactions for the entire byte count or a Split Completion Message indicating an error occurred as described in Section 2.10.6.2. A completer that executes Split Transactions for multiple Sequences concurrently is permitted to execute the transaction and initiate the Split Completions for different Sequences in any order. (Split Completions for the same Sequence must be initiated in address order.) As in conventional PCI, if a requester requires one non-posted transaction to complete before another, it must not initiate the second transaction until the first one completes.



## IMPLEMENTATION NOTE

### Mixing Immediate Response and Split Response

This note uses the following two terms to explain the limitations when Immediate Transactions and Split Transactions are used between a single pair of ADBs:

- ☐ Immediate-capable: an area of the device's address space that is capable of responding within the latency requirements defined in Section 2.9.1 such that the device executes read transactions as Immediate Transactions.
- ☐ Split-only: an area of the device's address space that cannot respond to read transactions within the latency requirements defined in Section 2.9.1 and thus the device must execute them as Split Transactions.

Device designers should use extreme care in mixing immediate-capable and split-only address spaces. Unless the address map is carefully laid out, the device must either provide data from the immediate-capable range in a Split Completion or must signal Target-Abort to some read transactions that would otherwise be legal.

Although requesters are generally required to understand the range limitations of the devices they address, completers have no ability to regulate the type and length of read commands that they are addressed by. Therefore the completer must respond to *any* read command within its address space. If the device is not designed to provide read data up to the next ADB as an Immediate Transaction, and the device is not designed to deliver all the data as a Split Transaction (as defined in Section 2.11.2.4), the device would have to signal Target-Abort to that read transaction and risk that the system will halt execution (see Section 2.11.2.5).

**Case 1:** When an immediate-capable address space precedes a split-only address space and both spaces are between a single pair of adjacent ADBs, the device is forced either to support providing the immediate-capable data within a Split Completion, or to signal Target-Abort for reads that cross those internal boundaries.

*Example 1:* A device with 128 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-3Fh for its immediate-capable register set and offsets 40h-7Fh as a window into some split-only memory. The device receives a memory read transaction for offset 00h with a byte count of 80h. Since the device cannot provide an immediate completion for offsets 40h-7Fh, the device cannot signal Data Transfer or Disconnect at Next ADB. Signaling either of these allows the device to disconnect the transaction no sooner than the next ADB, which is offset 80h. The device also cannot signal Single Data Phase Disconnect. Signaling Single Data Phase Disconnect for any read transaction that includes a split-only location is prohibited. (See Section 2.11.2.1.) The device is permitted to signal Single Data Phase Disconnect *only* if the read request is entirely contained within the offsets 00h-3Fh. If the read transaction includes any portion of the split-only range, the device must either signal Split Response and provide all 128 bytes of valid data in a single Split Completion or signal Target-Abort (and risk that the system will halt execution).

*Example 2:* A device with 256 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-7Fh for its immediate-capable register set and offsets 80h-FFh as a window into some split-only memory. The device receives a memory read transaction for offset 00h with a byte-count of 100h. Since the boundary between immediate-capable and split-only address spaces is located at an ADB (80h), the device is free to complete the transaction as an Immediate Transaction up to the ADB (signal Data Transfer, or Disconnect at Next ADB, or Single Data Phase Disconnect) and signal Split Response when the initiator continues the Sequence at the ADB.

**Case 2:** Whenever immediate-capable address space is located directly following split-only address space, the device is forced either to support providing the immediate-capable data within a Split Completion, or to signal Target-Abort to reads that cross those internal boundaries. This restriction applies even if the boundary is located at an ADB.

*Example 3:* A device with 128 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-3Fh as a window into some split-only memory and offsets 40h-7Fh for its immediate-capable register set. The device receives a memory read transaction for offset 00h with a byte count of 80h. Since the device is not permitted to provide a Split Completion with less than the requested byte count (see Section 2.10.2), it must either signal Split Response and provide all 128 bytes of valid data in a single Split Completion or signal Target-Abort (and risk that the system will halt execution).

*Example 4:* A device with 256 bytes of memory space assigned through a Base Address register chooses to use offsets 00h-7Fh as a window into some split-only memory and offsets 80h-FFh for its immediate-capable register set. The device receives a memory read transaction for offset 00h with a byte-count of 100h. Although the device could conceivably signal Split Response and then disconnect its Split Completion at the ADB, the device is still required to satisfy the entire byte count (see Section 2.10.2). Therefore, as in Example 3, the device must either signal Split Response and provide all 256 bytes of valid data in a Split Completion or signal Target-Abort.

## 2.10.2. Split Completion

A Split Completion transaction is a transaction that uses the Split Completion command. A Split Completion is a burst transaction, even if the Split Request was a DWORD transaction. As for all burst transactions, Split Completions include the byte count in the attribute phase. In PCI-X Mode 1, the C/BE# bus is reserved and driven high during all data phases of a Split Completion. In PCI-X Mode 2, Split Completion transactions are source-synchronous regardless of the byte count of the transaction, and even if the Split Request was a DWORD transaction. In PCI-X Mode 2, the C/BE# bus carries the data strobes for each subphase of a Split Completion (see Section 2.1.3.1, “Source-Synchronous Data Strobes,” in PCI-X EM 2.0).

Split Completion transactions address their targets differently than other burst transactions. The target of a Split Completion is the requester that initiated the Split Request. The completer stores the Requester ID (bus number, device number, and function number of the requester) from the attribute phase of the Split Request. The Requester ID becomes part of the Split Completion address driven on the AD bus during the address phase of the Split Completion. (See Section 2.10.3 for a complete description of the Split Completion address.) PCI-X bridges use the Requester ID to determine which transactions to forward. The requester uses the Requester ID to recognize Split Completions that correspond to its Split Requests.

The attributes of a Split Completion differ from the attributes of other burst transactions in that they carry information about the completer rather than the requester. Completer Attributes are specified in Section 2.10.4.

If the Split Request was a burst read and the completer does not encounter an error condition, the Split Completion includes read data and has one or more data phases, up to that required to satisfy the byte count of the Split Request. (See Section 2.10.6 for error conditions and Split Completion Messages.) The completer and intervening bridges are permitted to disconnect the Split Completion transaction on any ADB, following the same protocol as other burst transactions. Each time the completer resumes the Split Completion after an initiator or target disconnection, the address and byte count must be adjusted to the portion remaining in the Sequence. The completer must initiate all Split Completions resulting from a single Split Request (i.e., with the same Sequence ID) in address order. An intervening bridge must maintain the order of Split Completion transactions with the same Sequence ID (that is, it must keep them in address order).

If the completer intends to disconnect the Split Completion on the first ADB (i.e., the next higher ADB from the starting address of the Split Request), the completer is permitted to use a byte count smaller than that of the Split Request (see Section 2.10.4 for the Byte Count Modified bit requirements). (Note that this is the only way the completer can disconnect the transaction on an ADB that is closer than four data phases from the starting address. See Section 2.11.1.1.) The completer must never use a byte count other than the full remaining byte count that would stop the transaction anywhere other than the first ADB of the Sequence. As with all Split Completions, the completer must keep the Split Completion data in address order, even when changing the byte count to disconnect on the first ADB.

The completer is further restricted from using a byte count less than the full remaining byte count of the Sequence if both of the following are true:

- ❑ The device is designed to complete as an Immediate Transaction a burst memory read transaction to an address greater than or equal to one particular ADB,  $ADB_n$ , and less than the next higher ADB,  $ADB_{n+1}$ .
- ❑ The starting address of the Sequence being completed as a Split Transaction is  $ADB_{n+1}$ .

This limitation exists because in some cases the burst memory read Sequence has crossed a PCI-X bridge and what appears as the starting address to the completer is actually a continuation by the bridge of a larger Sequence. (See Section 8.4.2.2.) Completers that modify the byte count only when the starting address is not equal to an ADB automatically meet this requirement.

If the Split Request was a DWORD transaction, the Lower Address field in the Split Completion address (see Section 2.10.3) is set to zero and the Byte Count field in the Completer Attributes (see Section 2.10.4) is set to four, regardless of which byte enables were asserted in the Split Request. If the Split Request was a read transaction, data is driven on  $AD[31:00]$  during the data phase of the Split Completion (16 bits of the AD bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22). In PCI-X Mode 1, only byte lanes corresponding to the enabled bytes in the Split Request contain valid data. The requester must ignore the data in the other byte lanes (except for parity checking). Since the Lower Address is set to zero regardless of the address of the Split Request, in PCI-X Mode 2, data is valid only in the first subphase (two subphases on a 16-bit bus) and only in the byte lanes corresponding to the enabled bytes in the Split Request. The requester must ignore the data in the other byte lanes and subphases (except for ECC checking, see Section 5.1.2). If the Split Request was a write transaction, a Split Completion Message is driven on  $AD[31:00]$  (16 bits of the AD bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22) regardless of the byte enables asserted in the Split Request. See Section 2.10.6 for a complete description of Split Completion Messages.



## IMPLEMENTATION NOTE

### Starting Addresses and Byte Count for Split Completions

Split Completion transactions are burst transactions even if the corresponding Split Request was a DWORD transaction. The way the completer generates the starting address and byte count for a Split Completion varies according to the characteristics of the Split Request and Split Completion.

When a completer generates a Split Completion for a burst Split Request, it normally copies the lower seven bits of the starting address and the byte count from the Split Request to the Split Completion. If the completer intends to disconnect the Split Completion on the first ADB, it is permitted to use a byte count other than that of the Split Request and must set the Byte Count Modified bit in the Completer Attributes.

In the following cases, the Split Completion is a single DWORD, and the completer sets the Lower Address field in the Split Completion address to zero, and sets the Byte Count field in the Completer Attributes to four (without setting the Byte Count Modified bit), regardless of the size of the Split Request:

- ☐ The Split Request was a DWORD transaction.
- ☐ The Split Completion contains a Split Completion Message.

Like all burst transactions on 64-bit buses, Split Completions are permitted to be initiated as 64- or 32-bit transfers. That is, REQ64# is permitted to be either asserted or deasserted. The completer (or an intervening bridge) is permitted to initiate the Split Completion at either transfer width, regardless of the width or type of the Split Request. The width of each transaction is negotiated independent of all previous transactions in the Sequence and independent of the Split Request. (See Section 2.12.1.3.) For example, a 64-bit PCI-X bridge is permitted to initiate the Split Completion with REQ64# asserted even if the requester initiated the Split Request with REQ64# deasserted. (In this case, the requester would likely not assert ACK64#, so the Split Completion would proceed as a 32-bit transaction.) Furthermore, the completer is permitted to initiate the Split Completion as a 64-bit transfer (REQ64# asserted) even if the Split Request was a DWORD transaction. In this case, the completer would drive the DWORD of read data or the Split Completion Message on AD[31::00], since the starting address of the Split Completion (i.e., the Lower Address field of the Split Completion address) is set to 0 for completion of DWORD Split Requests. In other words, the transfer width and byte-lane requirements for a Split Completion are exactly the same as for a memory write of the identical length.

A completer is permitted to accept a single Split Request at a time. Such a device is permitted to terminate subsequent splittable transactions with Retry until the requester accepts the Split Completion. (Overall system performance is generally better if completers accept multiple Split Transactions at the same time.)

### 2.10.3. Split Completion Address

The Split Completion address is driven on the AD bus during the address phase of Split Completion transactions. The completer copies all this information from the address and attribute phases of the Split Request.

Figure 2-64 shows the bit assignments for the Split Completion address. The Split Completion command is driven on C/BE[3::0]# (two bits of the C/BE# bus during each of the two data phases if the bus is 16 bits wide, see Table 2-22). Table 2-13 describes the bit definitions of the Split Completion address fields.

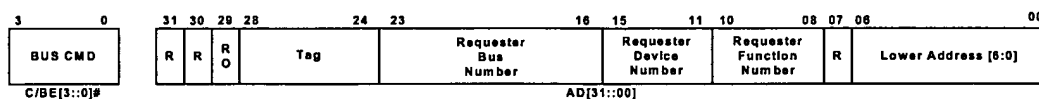


Figure 2-64: Split Completion Address

Table 2-13: Split Completion Address Field Definitions

Field	Function
Reserved (R)	Must be set to 0 by the initiator and ignored by the target (except for parity checking). PCI-X bridges forwarding a Split Completion must also set these bits to zero, even if they are set for the Split Completion received by the bridge. Future versions of the PCI-X definition may define these bits for additional features. PCI-X bridges designed to the present revision do not support such additional features and must set the bits to 0.
Relaxed Ordering (RO)	The completer copies this bit from the corresponding bit of the Requester Attributes (see Figure 2-1). Bridges throughout the system optionally use the bit to influence transaction ordering. A PCI-X bridge forwarding the Split Completion to another bus operating in PCI-X mode forwards this bit unmodified with the transaction, even if the bit is not used by the bridge.
Tag	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.
Requester Bus Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. A PCI-X bridge uses this field to identify transactions to forward. If this field of a Split Completion on the secondary bus is not between the bridge's secondary bus number and subordinate bus number, inclusive, and the primary interface is operating in PCI-X mode, the bridge forwards the transaction upstream. If this field of a Split Completion on the primary bus is between the bridge's secondary bus number and subordinate bus number, inclusive, and the secondary interface is operating in PCI-X mode, the bridge forwards the transaction downstream. If the bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified. See Section 8.4.3.1 for the use of this field by PCI-X bridges when one of the interfaces is operating in conventional mode.
Requester Device Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.
Requester Function Number	The completer copies this field from the corresponding field of the Requester Attributes. The requester uses this information to identify the appropriate Split Completions. If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it leaves this field unmodified.

Field	Function
Lower Address	<p>The completer copies this field from the least significant seven bits of the address of the Split Request, regardless of the command used by the Split Request, if all of the following are true:</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> The Split Request for this Sequence was a burst read.</li> <li><input type="checkbox"/> This is the first Split Completion of the Sequence.</li> <li><input type="checkbox"/> The Split Completion is not a Split Completion Message.</li> </ul> <p>If the Split Completion is disconnected on an ADB, this field is zero when the Sequence resumes.</p> <p>If the Split Request was a DWORD transaction or the Split Completion is a Split Completion Message, this field is set to zero.</p> <p>If a PCI-X bridge forwards the Split Completion to another bus operating in PCI-X mode, it uses this information to determine where the Split Completion starts relative to an ADB. The bridge leaves this field unmodified.</p>

## 2.10.4. Completer Attributes

The attribute phase of a Split Completion contains the Completer Attributes. The Completer Attributes are a combination of the Completer ID and information about the Sequence stored from the Split Request. Figure 2-65 shows the bit assignments for the Completer Attributes, and Table 2-14 describes the bit definitions.

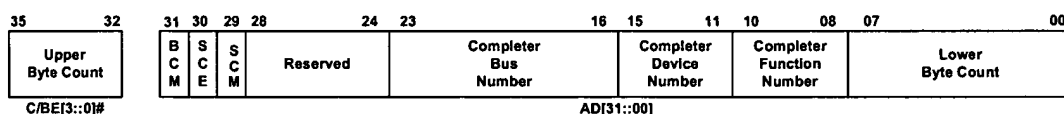


Figure 2-65: Completer Attribute Bit Assignments

Table 2-14: Completer Attribute Field Definitions

Attribute	Function															
Byte Count Modified (BCM)	<p>The completer must set this bit to 1 if the Byte Count field for this Split Completion contains a number smaller than the full remaining byte count of the transaction. (The completer is allowed to modify the byte count only to disconnect the transaction on the first ADB of the Sequence. See Section 2.10.2 for additional restrictions.) If the byte count field contains the full remaining byte count of the Split Request, or if the Split Completion is a Split Completion Message, the completer sets this bit to 0.</p> <p>This bit is used only for Split Completions resulting from burst read transactions (Memory Read Block and Alias to Memory Read Block) and is set to 0 for Split Completions resulting from all other commands.</p> <p>This bit is used for diagnostic purposes. Targets (bridges and requesters) are permitted to ignore this bit.</p>															
Split Completion Error (SCE)	<p>The completer sets this bit if the transaction is a Split Completion Message that is an error message (i.e., Message Class 1h or 2h). Requesters are permitted to use this information to differentiate between normal and error write completion messages before the actual message is latched and decoded. See the Split Completion Message attribute bit described below for additional requirements.</p>															
Split Completion Message (SCM)	<p>The completer sets this bit to 0 if the Split Completion contains read data. It sets this bit to 1 if the Split Completion contains a Split Completion Message. See Section 2.10.6 for a complete discussion of Split Completion Messages.</p> <p>The Split Completion Error and Split Completion Message bits are allowed together as follows:</p> <table><tr><th>SCE</th><th>SCM</th><th>Case</th></tr><tr><td>0</td><td>0</td><td>Normal completion of read (includes read data)</td></tr><tr><td>0</td><td>1</td><td>Normal completion of write (includes message)</td></tr><tr><td>1</td><td>0</td><td>reserved</td></tr><tr><td>1</td><td>1</td><td>Error completion (read or write, includes message)</td></tr></table>	SCE	SCM	Case	0	0	Normal completion of read (includes read data)	0	1	Normal completion of write (includes message)	1	0	reserved	1	1	Error completion (read or write, includes message)
SCE	SCM	Case														
0	0	Normal completion of read (includes read data)														
0	1	Normal completion of write (includes message)														
1	0	reserved														
1	1	Error completion (read or write, includes message)														
Reserved (R)	<p>Must be set to 0 by the completer and ignored by the requester (except for parity checking). PCI-X bridges forward these bits unmodified.</p>															
Completer Bus Number	<p>This 8-bit field identifies the completer's bus number. Completers supply this number from the Bus Number register in the PCI-X Status register. The value FFh is reserved and means the completer's PCI-X Status register has not been initialized.</p> <p>This information is used for diagnostic purposes on the bus.</p> <p>The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.</p>															

Attribute	Function
Completer Device Number	<p>This 5-bit field contains the device number assigned to the completer. Completers supply this number from the Device Number register in the PCI-X Status register. The value 1Fh is reserved and means the completer's PCI-X Status register has not been initialized. The Device Number of the source bridge is always 00h.</p> <p>The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.</p>
Completer Function Number	<p>This 3-bit field contains the function number of the completer within the device. This is the function number in the configuration address to which the function responds. Unlike the Device Number and Bus Number fields in the PCI-X Status register, the value of the Function Number field is assigned to the function by design and needs no initialization.</p> <p>The combination of the Completer Bus Number, Completer Device Number, and Completer Function Number is referred to as the Completer ID.</p>
Upper Byte Count, Lower Byte Count	<p>This 12-bit field is divided between the Upper Byte Count in the C/BE[3::0]# bus and the Lower Byte Count in the AD[7::0] bus. The target (requester or bridge) uses this information to determine the end of the transaction, particularly if the Split Completion has less than four data phases. In some cases, the completer copies this field from the corresponding field in the Requester Attributes. In other cases, the completer generates the value for this field. (See Section 2.10.2 for details.)</p> <p>There is no guarantee that the initiator will successfully move the entire byte count in a single transaction. If the Split Completion transaction is disconnected for any reason, the initiator must adjust the contents of the Byte Count field in the subsequent transactions of the same Sequence to be the number of bytes remaining in this Sequence.</p> <p>If the Split Completion is a Split Completion Message, the completer sets the byte count to four (see Section 2.10.6).</p> <p>The Byte Count is specified as a binary number, with 0000 0000 0001b indicating 1 byte, 1111 1111 1111b indicating 4095 bytes, and 0000 0000 0000b indicating 4096 bytes.</p>



## IMPLEMENTATION NOTE

### Use of the Byte Count Modified Attribute

The purpose of the Byte Count Modified bit in the Completer Attributes is to provide visibility for devices that monitor but do not participate in the bus protocol (such as a bus analyzer). PCI-X devices and bridges are not required to decode this bit.

For example, suppose a 64-bit requester initiates a Memory Read Block transaction for 280 bytes starting at address 104 (three data phases from the ADB at address 128). The

completer (on the same bus) signals Split Response and fetches the data. Further suppose that the completer wants to disconnect the Split Completion transaction at the first ADB (address 128) and, therefore, changes the byte count in the Completer Attributes to 24 bytes and sets the Byte Count Modified bit to 1. A logic analyzer monitoring the bus observes the Byte Count Modified bit set and realizes that the Sequence is not complete, even though the byte count of 24 is satisfied. After the first split completion, the completer regains bus ownership, issues a new Split Completion with a byte count of 256 (the remaining byte count), and sets the Byte Count Modified bit to 0.

---

### 2.10.5. Requirements for Accepting Split Completions

The requester is required to accept all Split Completions resulting from its own Split Requests. That is, the requester is required to assert **DEVSEL#** on all Split Completions in which the Sequence ID (Requester ID and Tag) corresponds to a Split Request issued by that device. See Section 5.2.5 for Split Completions that are unexpected or corrupted.

If the requester asserts **DEVSEL#** for a Split Completion, the requester must accept the entire byte count requested without signaling Split Response, Retry, Single Data Phase Disconnect, or Disconnect at Next ADB. If the requester no longer needs the Split Completion data, the requester must accept it and then discard it. In general, a requester must have a buffer ready to receive the entire byte count for all Split Requests it issues.

The requester is permitted to signal Target-Abort for a Split Completion only under error conditions in which the integrity of data in the system cannot be guaranteed. An example of such an error condition is an uncorrectable error in the Split Completion address, which includes the Sequence ID. (See Section 5.2.3 for requirements for the target also to assert **SERR#** in this case.) In some cases, signaling Target-Abort for a Split Completion causes another device to assert **SERR#**. (See Sections 0 and 8.7.1.3.) The requester must assume that a possible consequence of signaling Target-Abort for a Split Completion transaction is that the system will halt execution.

Bridges (PCI-X bridges and application bridges) are permitted to terminate Split Completions with Retry and to disconnect multi-data-phase Split Completions in some cases. See Section 8.4.5 for more details.

If a requester issues more than one Split Request at a time (with different Tags), the requester must accept the Split Completions from the separate requests in any order. (Split Completions with the *same* Tag originate from the same Split Request and always arrive in address order.)

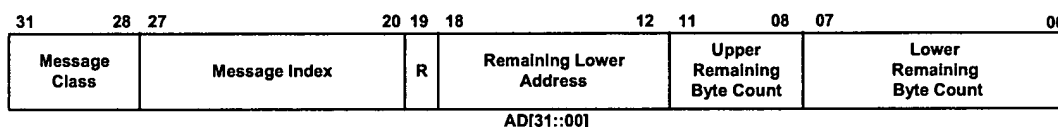
### 2.10.6. Split Completion Messages

If the SCM bit in the Completer Attributes is set, the transaction includes a message. Split Completion Messages notify the requester when a split write request (I/O or configuration) has completed, and they indicate error conditions in which delivery of data for a read request or execution of a write request is not possible.

A Split Completion Message is a burst transaction (like all Split Completions) but is always a single DWORD in length regardless of the size and type of the Split Request. The Lower Address field in the Split Completion address is set to zero, and the Byte Count field in the Completer Attributes is set to four for all Split Completion Messages. In PCI-X Mode 1, the C/BE# bus is reserved and driven high during the data phase of a Split Completion Message as it is for all Split Completions. Note that in PCI-X Mode 2, the C/BE# bus carries data strobes for each subphase of the Split Completion (see Section 2.1.3.1, “Source-Synchronous Data Strobes,” in PCI-X EM 2.0), and the Split Completion Message appears in the first subphase, as would be true for any Split Completion in which the Lower Address bits were 0 and the byte count was four.

A Split Completion Message terminates a Sequence regardless of how many bytes remain to be sent. If the Split Request was a burst read, the Byte Count field in the Split Completion Message indicates the number of bytes that were not sent for this Sequence, and the Remaining Lower Address field indicates the lower seven bits of the starting address of the remainder of the Sequence. (PCI-X bridges use this information to release buffer space that was reserved by the Split Request.)

Figure 2-66 shows the format of the message in the data phase of the Split Completion, and Table 2-15 shows the encoding of those messages.



**Figure 2-66: Split Completion Message Format**

**Table 2-15: Split Completion Message Fields**

Field	Function
Message Class	Split Completion Messages are in one of the following classes. All other values are reserved. <div style="margin-left: 40px;"> 0h      Write Completion (See Section 2.10.6.1.)  1h      PCI-X Bridge Error (See Section 8.8.)  2h      Completer Error (See Section 2.10.6.2.) </div>
Message Index	Identifies the type of message within the message class. See Table 2-16 and Table 2-17.
Reserved (R)	Must be set to 0 by the completer (or intervening bridge) and ignored by the requester (or intervening bridge).
Remaining Lower Address	If the Split Request was a burst memory read, this field contains the least significant seven bits of the address of the first byte of read data that has not previously been sent. If the Split Request was a DWORD transaction, the completer sets this field to zero. PCI-X bridges use this number to manage buffer space reserved for Split Completions.

Field	Function
Upper and Lower Remaining Byte Count	If the Split Request was a burst memory read, the completer sets this field to the number of bytes of read data that have not previously been sent. If the Split Request was a DWORD transaction, the completer sets this field to 4. PCI-X bridges use this number to manage buffer space reserved for Split Completions.

### 2.10.6.1. Write Completion Message Class

The Write Completion class is used for Split Write Completion messages. The Remaining Lower Address field is set to zero and the Upper and Lower Remaining Byte Count field set to four in the data phase of this Split Completion Message (PCI-X bridges reserve a single DWORD for a Split Write Completion). (The Lower Address in the Split Completion address is also zero and the byte count in the Completer Attributes is also four, as it is for all Split Completion Messages.)

Only one message index is defined in this class, as shown in Table 2-16. All other indices are reserved.

Table 2-16: Write Completion Message Index (Class 0)

Index	Message
00h	Normal completion

### 2.10.6.2. Completer Error Message Class

After signaling Split Response, if the completer encounters an abnormal condition that prevents it from executing a Split Transaction, the completer must notify the requester of the abnormal condition by sending a Split Completion Message with the Completer Error class. Examples of such conditions include the following:

1. The byte count of the request exceeds the range of the completer.
2. Parity errors internal to the completer.

If the byte count exceeds the range of the completer, the completer must initiate Split Completion transactions with read data up to the device boundary and then disconnect the Sequence. The completer then terminates the Sequence by sending the Split Completion Message. In all other cases, the completer is permitted to send a Split Completion Message of this class in lieu of the first Split Completion, or any continuation in a Sequence (after a disconnection), independent of the actual address of the error in the Sequence. The only inference that can be made as to the actual address of the error is as follows:

1. If the Sequence was previously disconnected on an ADB, the address of the error is greater than the last address of the Split Completion transactions that were previously sent without error for this Sequence.
2. The error address is less than or equal to the ending address of the Sequence.

Table 2-17 shows the index values defined for this message class. All other indices are reserved.

**Table 2-17: Completer Error Messages Indices (Class 2)**

Index	Message
00h	<p>Byte Count Out of Range.</p> <p>The completer uses this message if the sum of the address and the byte count of the Split Request exceeds the address range of the completer.</p> <p>The completer must initiate Split Completion transactions with read data up to the device boundary.</p> <p>A normally functioning requester understands the address range of the completer it is attempting to read and does not request data that is out of range. The completer sends this message to indicate to the requester the occurrence of an error condition. (The error could have occurred either in the completer or in the requester). The requester must report this error condition to its device driver.</p>
01h	<p>Uncorrectable Split Write Data Error.</p> <p>The completer sends this message if it terminated a DWORD write transaction with Split Response and detected an uncorrectable data error. (See Section 5.2.4.)</p>
8Xh	<p>Device-Specific Error.</p> <p>The completer uses this message if it encounters an error that prevents execution of the Split Request, and the error is not indicated by one of the other error messages. The lower four bits of the index are available for the device to encode device-specific error or diagnostic information. The vendor of the device must provide documentation for this field.</p>



## IMPLEMENTATION NOTE

### Reporting Device-Specific Error Messages

One way to report the receipt of a device-specific error Split Completion Message is for the requester to store the lower four bits of the message index in a device-specific location and cause an interrupt to the processor. The device driver servicing the interrupt would read the register.

A common practice in cases such as these is to reserve one error encoding (e.g., 0h) to indicate no error condition. In this case, software would clear the register after the occurrence of each error, so it could tell the difference between new errors and errors it had already recorded.

Alternatively, all 16 codes could be assigned to different errors and an additional device-specific bit assigned to indicate that the register contains a new error condition.

## 2.11. Transaction Termination

The figures in this section show the methods by which transactions are terminated. Those methods include the following:

- ❑ Initiator Termination
  - Initiator Disconnection or Satisfaction of Byte Count
  - Master-Abort Termination
- ❑ Target Termination
  - Single Data Phase Disconnection
  - Disconnection at Next ADB
  - Retry Termination
  - Split Response Termination
  - Target-Abort Termination

Transaction termination rules are based on number of data phases and are independent of the width of the transaction (64, 32, or 16 bits). That is, the amount of data transferred in each data phase varies, but the rules based on number of data phases are the same.

Most of the figures in this section that illustrate transaction termination show **DEVSEL#** decode A and no target initial wait states. Decode speed A is not allowed in ECC mode. All other **DEVSEL#** decodes and wait state combinations specified in Sections 2.8 and 2.9.1 are allowed.

In PCI-X Mode 1, when a transaction ends, the initiator deasserts and floats **FRAME#** as described in PCI 2.3 for sustained tri-state signals, that is, the initiator actively deasserts **FRAME#** for one clock and then floats it. For those PCI-X transactions that contain one or two data phases (i.e., when **FRAME#** deasserts at the same time **IRDY#** deasserts), this timing is required to avoid conflicts with the next bus owner (e.g., see Figure 2-69 and Figure 2-70). However, for those transactions that contain three, four, or more data phases (i.e., when **FRAME#** deasserts before **IRDY#** deasserts) the initiator optionally deasserts **FRAME#** for one clock and then floats it (as in the one- and two- data phase cases) or deasserts **FRAME#** for two clocks and then floats it. Most of the figures in this section that show three, four, or more data phases, show **FRAME#** deasserted for two clocks before floating. Figure 2-68 illustrates the other alternative.

In PCI-X Mode 2, the assertion, deassertion, and float requirements for control signals related to transaction termination are the same as in PCI-X Mode 1 except that there are a minimum of two idle clocks (also described as one idle clock and a bus turn-around clock) between two transactions. (In some cases in Mode 1, the next transaction begins after only one idle clock. See Section 4.2.1.)

Most of the figures in this section that illustrate transaction termination apply both to read and write transactions, and, therefore, do not show the AD bus. In all cases the initiator and target begin driving the AD and C/BE# buses as described in Sections 2.8 and 2.9 for the

appropriate DEVSEL# timing and number of wait states. In PCI-X Mode 1, the initiator and target float the AD and C/BE# buses according to the following rules:

1. Initiator:
  - a. If the transaction has four or more data phases, the initiator floats the C/BE# bus on the clock it deasserts IRDY#. If the transaction has less than four data phases, the initiator floats the C/BE# bus either on the clock it deasserts IRDY# or one clock after that.
  - b. If the transaction is a write with four or more data phases, the initiator floats the AD bus on the clock it deasserts IRDY#. If the transaction is a write with less than four data phases, the initiator floats the AD bus either on the clock it deasserts IRDY# or one clock after that.
2. Target: If the transaction is a read, the target floats the AD bus on the clock after the last data phase, regardless of the number of data phases in the transaction or the type of termination. That is, the target floats the AD bus on the clock it deasserts DEVSEL#, STOP#, and/or TRDY# after signaling the last Data Transfer or target termination.

## 2.11.1. Initiator Termination

All of the figures in this section illustrate transaction termination using PCI-X Mode 1. Transaction termination in PCI-X Mode 2 would be the same, except that in PCI-X Mode 2, the initiator and target drive and float the AD and C/BE# buses as described in Sections 2.16 and 4.1.2.

### 2.11.1.1. Initiator Disconnection or Satisfaction of Byte Count

Transactions that are disconnected by the initiator on an ADB before the byte count has been satisfied and those that terminate at the end of the byte count appear the same on the bus. Initiator termination of a transaction with four or more data phases differs from the case in which the transaction has less than four data phases.

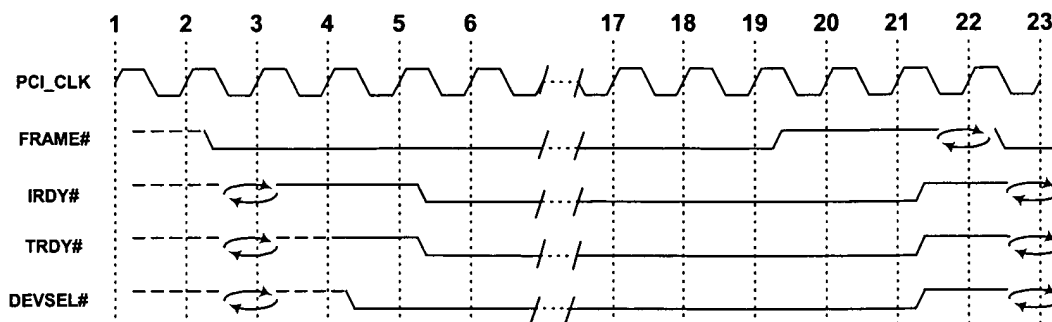


Figure 2-67: Initiator Termination of a Burst Transaction with Four or More Data Phases

Figure 2-67 illustrates initiator termination after four or more data phases. In this case, the initiator signals the end of the transaction by deasserting **FRAME#** one clock before the last data phase. It deasserts **IRDY#** on the clock after the last data phase.

Initiator termination in less than four data phases occurs only if the starting address, byte count, and width of the bus are such that the transaction has less than four data phases. If the initiator intends to disconnect a transaction on the first ADB, and the width of the bus is such that the starting address is less than four data phases from the ADB, the initiator must use a byte count that terminates the transaction on the ADB.

Table 2-18 shows the number of data phases for PCI-X Mode 1 and common-clock PCI-X Mode 2 transactions up to 12 DWORDs long. (For this table, the length of the transaction is measured from the starting address rounded down to the next DWORD address.) As the table shows, the number of data phases is equal to the number of DWORDs if the transaction width is 32 bits. If the transaction width is 64-bits, the number of data phases depends on whether the transaction starts on an even or odd DWORD. Data always transfers in an even number of data phases for common-clock Mode 2 transactions on a 16-bit bus.

**Table 2-18: Data Phases Dependence on Starting Address and Bus Width, Mode 1 and Common-Clock Mode 2**

Transaction Length (DWORDs)	Data Phases			
	64-bit Transfers		32-bit Transfers	16-Bit Transfers
	Starting on Even DWORD	Starting on Odd DWORD		
1	1	1	1	2
2	1	2	2	4
3	2	2	3	6
4	2	3	4	8
5	3	3	5	10
6	3	4	6	12
7	4	4	7	14
8	4	5	8	16
9	5	5	9	18
10	5	6	10	20
11	6	6	11	22
12	6	7	12	24

Table 2-19 and Table 2-20 show the number of data phases for PCI-X Mode 2 source-synchronous transactions up to 25 DWORDs long. (As in the previous table, the length of the transaction is measured from the starting address rounded down to the next DWORD address.) The headings under each transfer width indicate the DWORD address that contains the starting address. The DWORD address corresponds with **AD[2]**, **AD[3:2]**, or **AD[4:2]**, as indicated in the tables.

**Table 2-19: Data Phases Dependence on Starting Address and Bus Width,  
Source-Synchronous PCI-X 266**

Transaction Length (DWORDs)	Data Phases							
	64-Bit Transfer AD[3::2]				32-Bit Transfer AD[2]		16-Bit Transfers AD[2]	
	0	1	2	3	0	1	0	1
1	1	1	1	1	1	1	1	1
2	1	1	1	2	1	2	2	2
3	1	1	2	2	2	2	3	3
4	1	2	2	2	2	3	4	4
5	2	2	2	2	3	3	5	5
6	2	2	2	3	3	4	6	6
7	2	2	3	3	4	4	7	7
8	2	3	3	3	4	5	8	8
9	3	3	3	3	5	5	9	9
10	3	3	3	4	5	6	10	10
11	3	3	4	4	6	6	11	11
12	3	4	4	4	6	7	12	12
13	4	4	4	4	7	7	13	13
14	4	4	4	5	7	8	14	14
15	4	4	5	5	8	8	15	15
16	4	5	5	5	8	9	16	16
17	5	5	5	5	9	9	17	17
18	5	5	5	6	9	10	18	18
19	5	5	6	6	10	10	19	19
20	5	6	6	6	10	11	20	20
21	6	6	6	6	11	11	21	21
22	6	6	6	7	11	12	22	22
23	6	6	7	7	12	12	23	23
24	6	7	7	7	12	13	24	24
25	7	7	7	7	13	13	25	25

Table 2-20: Data Phases Dependence on Starting Address and Bus Width, Source-Synchronous PCI-X 533

Transaction Length (DWORDs)	Data Phases															
	64-Bit Transfer AD[4::2]								32-Bit Transfer AD[3::2]				16-Bit Transfer AD[3::2]			
	0	1	2	3	4	5	6	7	0	1	2	3	0	1	2	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	2	1	1	1	2	1	2	1	2
3	1	1	1	1	1	1	2	2	1	1	2	2	2	2	2	2
4	1	1	1	1	1	2	2	2	1	2	2	2	2	3	2	3
5	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3
6	1	1	1	2	2	2	2	2	2	2	2	3	3	4	3	4
7	1	1	2	2	2	2	2	2	2	2	3	3	4	4	4	4
8	1	2	2	2	2	2	2	2	2	3	3	3	4	5	4	5
9	2	2	2	2	2	2	2	2	3	3	3	3	5	5	5	5
10	2	2	2	2	2	2	2	3	3	3	3	4	5	6	5	6
11	2	2	2	2	2	2	3	3	3	3	4	4	6	6	6	6
12	2	2	2	2	2	3	3	3	3	4	4	4	6	7	6	7
13	2	2	2	2	3	3	3	3	4	4	4	4	7	7	7	7
14	2	2	2	3	3	3	3	3	4	4	4	5	7	8	7	8
15	2	2	3	3	3	3	3	3	4	4	5	5	8	8	8	8
16	2	3	3	3	3	3	3	3	4	5	5	5	8	9	8	9
17	3	3	3	3	3	3	3	3	5	5	5	5	9	9	9	9
18	3	3	3	3	3	3	3	4	5	5	5	6	9	10	9	10
19	3	3	3	3	3	3	4	4	5	5	6	6	10	10	10	10
20	3	3	3	3	3	4	4	4	5	6	6	6	10	11	10	11
21	3	3	3	3	4	4	4	4	6	6	6	6	11	11	11	11
22	3	3	3	4	4	4	4	4	6	6	6	7	11	12	11	12
23	3	3	4	4	4	4	4	4	6	6	7	7	12	12	12	12
24	3	4	4	4	4	4	4	4	6	7	7	7	12	13	12	13
25	4	4	4	4	4	4	4	4	7	7	7	7	13	13	13	13

Figure 2-68 through Figure 2-70 illustrate initiator termination after three, two, and one data phases, respectively. In each of these cases, the initiator deasserts **FRAME#** two clocks after the target asserts **TRDY#**. The initiator deasserts **IRDY#** one clock after the *last* data phase but never less than two clocks after the *first* data phase (the clock in which **FRAME#** is deasserted). The target deasserts **TRDY#** and **DEVSEL#** on the first clock after the byte count provided by the initiator is satisfied. Note that the three- and one-data phase cases described are not possible in 16-bit common-clock transactions because 16-bit common-clock transactions that transfer data always have an even number of data phases.

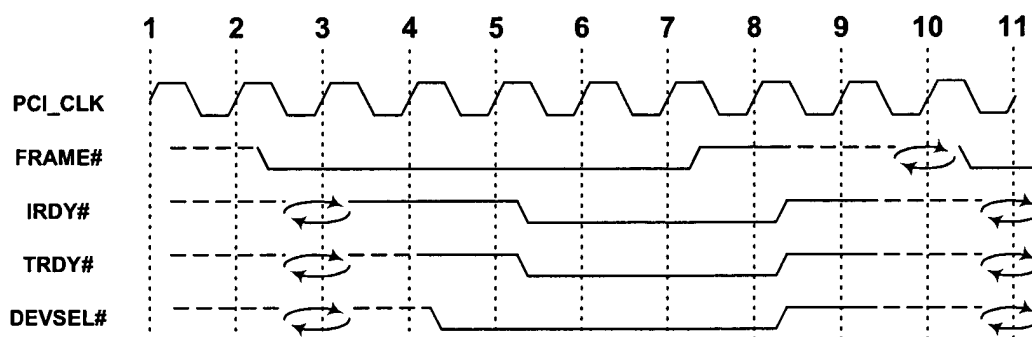


Figure 2-68: Initiator Termination of a Burst Transaction with Three Data Phases

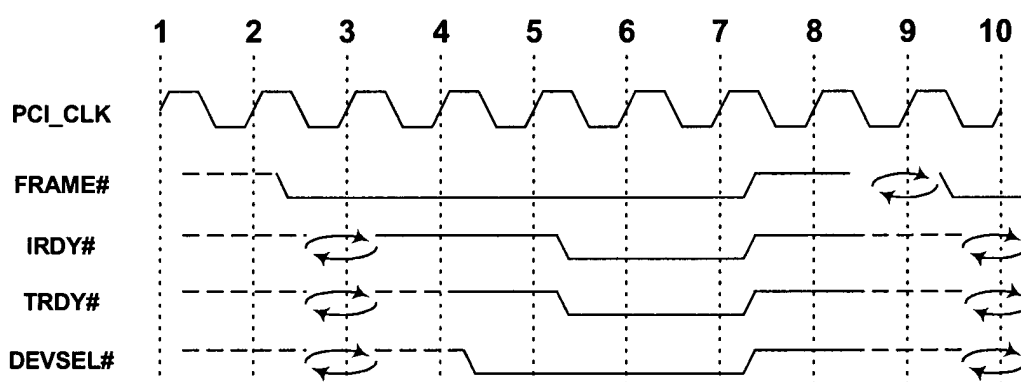


Figure 2-69: Initiator Termination of a Burst Transaction with Two Data Phases

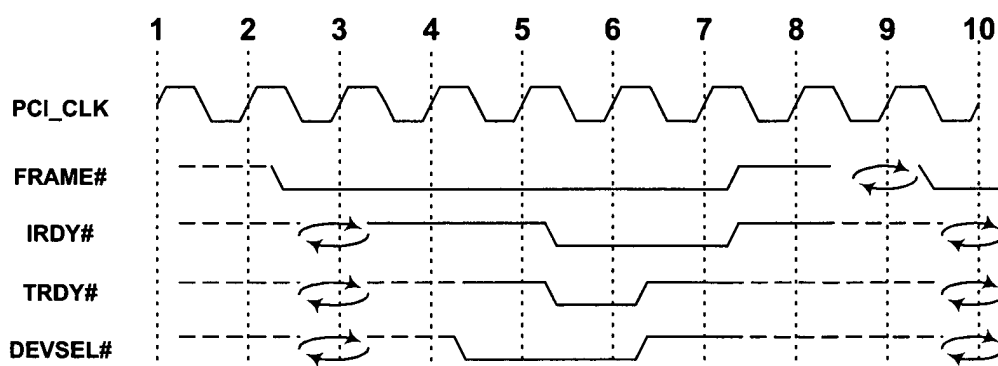


Figure 2-70: Initiator Termination of a Burst Transaction with One Data Phase

### 2.11.1.2. Master-Abort Termination

If no target asserts **DEVSEL#** within five clocks after the attribute phase (the second attribute phase in 16-bit transactions), the initiator deasserts **FRAME#** and **IRDY#** seven

clocks after the attribute phase(s). In Mode 1, the initiator floats the bus one clock later. In Mode 2, the initiator continues to drive the bus until a turn-around cycle (see Section 4.1.2). The initiator sets bits in its Status register the same as for conventional PCI.

Figure 2-71 shows a Master-Abort termination of a transaction.

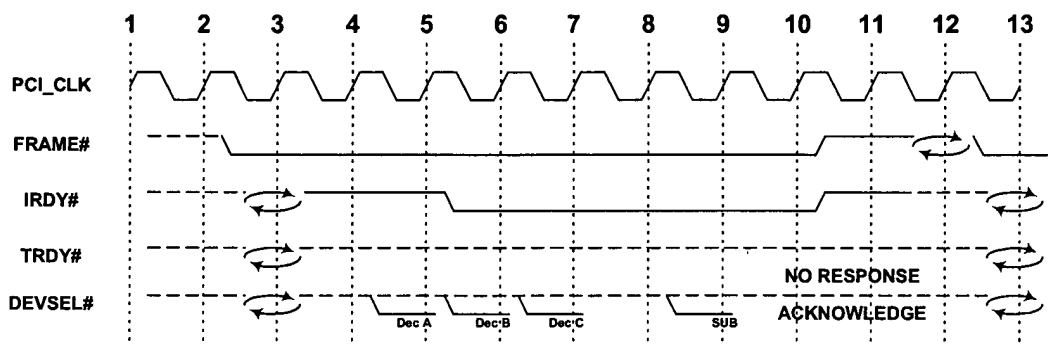


Figure 2-71: Master-Abort Termination

### 2.11.2. Target Termination and Data Phase Signaling

After a target asserts DEVSEL# in the target response phase, it must complete the transaction with one or more data phases. The target signals its intention on each clock after the target response phase with a combination of the target control signals, DEVSEL#, STOP#, and TRDY#. Table 2-21 shows all of the alternatives for target data phase signaling.

Table 2-21: Target Data Phase Signaling

Target Data Phase Signaling	DEVSEL#	STOP#	TRDY#	Data Transfer	Transaction Terminates or Continues
Master-Abort (Note 1)	Deassert	Deassert	Deassert	na	na
Split Response	Deassert	Deassert	Assert	(Note 2)	Terminates
Target-Abort	Deassert	Assert	Deassert	No	Terminates
Single Data Phase Disconnect	Deassert	Assert	Assert	Yes	Terminates
Wait State	Assert	Deassert	Deassert	No	Continues
Data Transfer	Assert	Deassert	Assert	Yes	Continues
Retry	Assert	Assert	Deassert	No	Terminates
Disconnect at Next ADB	Assert	Assert	Assert	Yes	(Note 3)

**Notes:**

- Shown for reference only. Not allowed after DEVSEL# is asserted. No target drives DEVSEL#, STOP#, and TRDY# for a transaction terminated with Master-Abort. The signals are deasserted by their respective pull-up resistors.

2. No data transfers on a Split Response for a read transaction. The target latches data on a Split Response for a write transaction. However, in both cases, the Sequence is not complete until the requester receives the Split Completion.
3. If the target signals Disconnect at Next ADB, the transaction continues to an ADB. (See Section 2.11.2.2 for details.)

A data phase ends each time the target signals anything other than Wait State (which is permitted only on the first data phase). See Section 2.9.1 for a discussion of the number of wait states permitted and the target initial latency. For DWORD transactions and common-clock burst transactions on a 16-bit bus, the target signals data phase action in pairs of clocks if it signals Data Transfer, Single Data Phase Disconnect, or Split Response. (16-bit common-clock transactions that transfer data always have an even number of data phases. See Section 2.12.2.3.)

If the target signals Data Transfer on one data phase, the transaction continues until the byte count is satisfied or the initiator terminates the transaction. The target is limited to signaling Data Transfer, Disconnect on Next ADB, or Target-Abort on subsequent data phases.

If the target signals Split Response, Target-Abort, Single Data Phase Disconnect, or Retry, the transaction terminates immediately. The transaction terminates on an ADB if the target signals Disconnect at Next ADB (see Section 2.11.2.2 for details).

When the transaction terminates (either by initiator or target termination), the target deasserts DEVSEL#, STOP#, and TRDY# one clock after the last data phase (if they are not already deasserted) and floats them one clock after that.

Targets must not store any information about a transaction after it is terminated either by the initiator or the target in any method other than Split Response. (Storing of transaction information for diagnostic purposes is permitted, if such information does not affect the device's response to transactions on the bus. Target-Abort termination and some error conditions require the target to set bits in the Status register.) Delayed Transactions are not permitted. For example, if a target collects data up to the byte count of a read transaction, delivers some of that data by signaling Data Transfer (i.e., executes it as an Immediate Transaction), and the transaction is disconnected (either by the target or the initiator), the target must discard the remainder of the data, unless the target guarantees that the buffered data will not become stale. The target must not assume that the initiator will continue any read operation after a transaction is terminated by the initiator or the target.

All of the figures in this section illustrate transaction termination using PCI-X Mode 1. Transaction termination in PCI-X Mode 2 would be the same, except that in PCI-X Mode 2, the initiator and target drive and float the AD and C/BE# buses as described in Sections 2.16 and 4.1.2, and for DWORD and common-clock burst transactions on a 16-bit bus the target signals Data Transfer, Single Data Phase Disconnect, or Split Response in pairs of clocks (see Section 2.12.2.3).

### **2.11.2.1. Single Data Phase Disconnection**

The target signals its intention to complete a single data phase (two data phases on a 16-bit common-clock transaction) and then disconnect the transaction by signaling Single Data Phase Disconnect. The target signals Single Data Phase Disconnect by asserting TRDY# and STOP# and deasserting DEVSEL# on the first data phase of the transaction (with or

without preceding wait states up to the maximum specified in Section 2.9.1). Signaling Single Data Phase Disconnect is not permitted on Split Completion or Device ID Message transactions. It is permitted on all other burst transactions (even if the byte count is small enough to limit the transaction to a single data phase) and DWORD transactions (which are always a single data phase). If the target signals Single Data Phase Disconnect, the transaction contains only a single data phase (two data phases on a 16-bit common-clock transaction), and the initiator deasserts **FRAME#** and **IRDY#** two clocks after the data phase (or first data phase on a 16-bit bus, see Section 2.12.2.3.3). In PCI-X Mode 2, if the transaction is a source-synchronous transaction, it contains a single data phase with all its subphases, but only the data of the subphase that includes the starting address is transferred. The other subphases are ignored, except for ECC checking (see Section 5.1.2).

Targets must be designed never to signal both Single Data Phase Disconnect and Data Transfer for memory write transactions that begin four or less data phases before any single ADB, unless the target verifies that the byte count is small enough not to include that ADB. That is, if a target is designed to signal Single Data Phase Disconnect for a memory write transaction with an address four or less data phases before an ADB, that target must be designed never to signal Data Transfer for a memory write transaction that begins four or less data phases from that same ADB and has a byte count large enough to include the ADB.

Targets are permitted to signal Single Data Phase Disconnect for a memory read transaction only if they are prepared to complete all transactions that are continuations of that Sequence, up to the next ADB, as Immediate Transactions. That is, the device must *not* signal Single Data Phase Disconnect for any individual read transaction of a Sequence if all of the following are true:

- ☐ The transaction is a burst read.
- ☐ The byte count is such that the transaction addresses at least one location to which the device will respond with Split Response when the Sequence is continued by the initiator.
- ☐ There is *not* an ADB between the first address of the transaction and the location to which the device will respond with Split Response when the Sequence is continued.

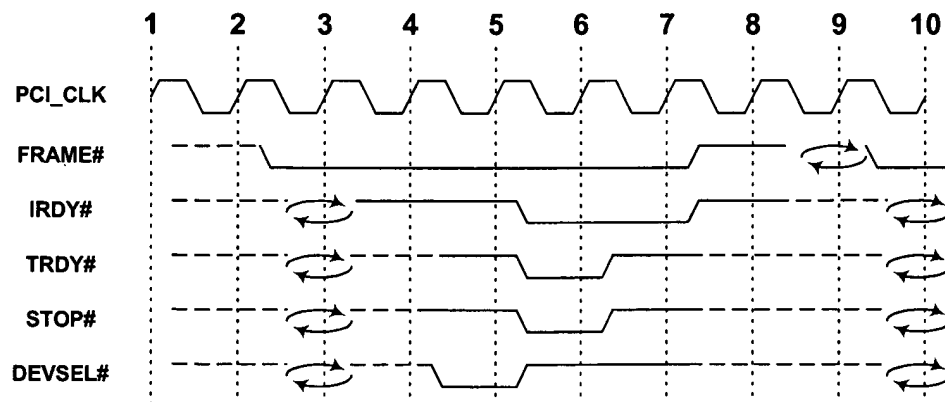


Figure 2-72: Single Data Phase Disconnection



## IMPLEMENTATION NOTE

### Use of Single Data Phase Disconnection

Single data phase disconnection is intended for address spaces such as control registers that generally are not accessed using burst transactions. Although it is permitted for both read and write transactions, its most common application is for writes. In some cases, memory write transactions that are intended to be separate transactions are combined into a single transaction by a host bridge or a conventional PCI bridge. The target avoids having to accept a burst up to the next ADB by signaling Single Data Phase Disconnect on each data phase. If the target signals Single Data Phase Disconnect for a location that is frequently addressed with multiple-data-phase burst transactions, the device's performance is severely reduced.

---



## IMPLEMENTATION NOTE

### Single Data Phase Disconnection and Memory Write Transactions

If a target signals Single Data Phase Disconnect for a memory write transaction that starts close to an ADB and signals Data Transfer for the continuation of that memory write, a PCI-X bridge will be unable to forward the memory write transaction in the following case:

1. A requester initiates a long memory write transaction (e.g., a host bridge combines many small memory write transactions) addressing a completer on the other side of a PCI-X bridge.
2. The bridge does not have buffer space available to hold the entire byte count of the memory write. However, it does have space for several ADQs of memory write data, so it responds to the transaction with Data Transfer and begins accepting data.
3. As the last bridge buffer fills, the bridge signals Disconnect at Next ADB, and the requester disconnects the transaction at the next ADB.
4. The bridge forwards this first memory write transaction of the Sequence to the destination bus.
5. The completer responds to the memory write transaction with Single Data Phase Disconnect and continues to do so for each continuation of the Sequence until the bridge holds less than four data phases of data in its buffers.

If the completer were to respond to the next continuation of the memory write Sequence with Data Transfer, the bridge would not be able to disconnect the transaction at the next ADB, because the continuation began less than four data phases from an ADB. The bridge could not continue beyond the ADB, because the requester has not yet written that data on the originating bus.

---



## IMPLEMENTATION NOTE

### Single Data Phase Disconnection and Burst Memory Read Transactions

The use of Single Data Phase Disconnect for burst memory read transactions is allowed, but rarely occurs in actual applications. In normal use, a device that is designed to respond with Single Data Phase Disconnect is never addressed by burst memory read transactions. Therefore, completers are limited in the way they respond to burst read transactions if they mix Single Data Phase Disconnect and Split Response between a single pair of adjacent ADBs.

For example, a device with 256 bytes of memory space assigned through a Base Address register is designed to respond with Split Response if address offset A0h is read. If the device is addressed by a read transaction starting at offset 00h with a length of 256 bytes, the device would be permitted to signal Single Data Phase Disconnect. In this case, there is an ADB (offset 80h) between the first address of the read transaction and the Split Response address. However, as the initiator continues reading from the disconnection point, the starting address eventually advances to the ADB (offset 80h) with a length of 128 bytes. In this case, the device would not be permitted to signal Single Data Phase Disconnect because there is no ADB between the starting address and the address to which the device will respond with Split Response (A0h).

If the same device is addressed by a different Sequence starting at address 80h with a byte count of 32 bytes (that is, an ending address of 9Fh), it is permitted to signal Single Data Phase Disconnect because the Sequence does not include any locations to which the device will respond with Split Response when the initiator resumes after the disconnection.

See Section 2.10.1 for additional design considerations when locations of this type are mixed between the same two ADBs.

This restriction on the use of Single Data Phase Disconnect and Split Response simplifies the design of PCI-X bridges forwarding a burst read transaction. Without this restriction a PCI-X bridge forwarding a burst read request would have to deal with the possibility that the completer could signal Single Data Phase Disconnect at the beginning of the transaction and then change to Split Response midway between two ADBs. A PCI-X bridge would not generally be able to create a Split Completion for the transactions if it only held a portion of the data that ended midway between two ADBs. The completer is permitted to change from an immediate completion to a Split Response only at an ADB.

---

#### 2.11.2.2. *Disconnection at Next ADB*

The target signals its intention to disconnect the transaction at the next ADB by signaling Disconnect at Next ADB. The target signals Disconnect at Next ADB by asserting TRDY#, DEVSEL#, and STOP# on any data phase of the transaction. The target is permitted to signal Disconnect at Next ADB regardless of the starting address or length of the transaction, or whether the transaction is a burst or DWORD. Some restrictions apply to

the use of Disconnect at Next ADB by bridges (see Section 8.4.6). Once the target has signaled Disconnect at Next ADB, it is limited to signaling Disconnect at Next ADB or Target-Abort on all subsequent data phases until the end of the transaction. (The transaction ends immediately after the target signals Target-Abort. See Section 2.11.2.5.)

If the length of a transaction is such that it does not cross the next ADB (i.e., if it is a DWORD transaction or the byte count of a burst is satisfied before reaching the next ADB), Disconnect at Next ADB is treated by the initiator the same as Data Transfer. If the transaction is a burst that would otherwise cross the next ADB and the target signals Disconnect at Next ADB on the first data phase of the transaction, the transaction ends at the first ADB. If the target signals Disconnect at Next ADB after the first data phase and four or more data phases before an ADB, the initiator disconnects the transaction on that ADB. If the target signals Disconnect at Next ADB after the first data phase and less than four data phases before an ADB, the transaction crosses that ADB and continues to the next ADB (unless the byte count is satisfied before that).

The following figures illustrate Disconnect at Next ADB. In these illustrations, the number of clocks between ADBs is calculated for a common-clock 64-bit burst transfer. The number of clocks is different for other combinations of transfer widths and modes. (See Table 2-1.) Figure 2-73 illustrates Disconnect at Next ADB after the first data phase and four data phases from an ADB.

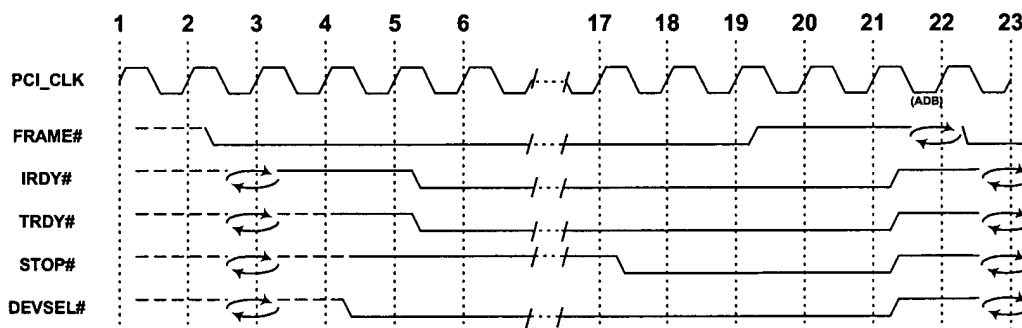


Figure 2-73: Disconnect at Next ADB Four Data Phases from an ADB

Figure 2-74 illustrates the case in which the target signals Disconnect at Next ADB on various data phases relative to an ADB. If the target signals Disconnect at Next ADB after the first data phase and less than four data phases from an ADB (clocks 7, 8, or 9 in the figure), the transaction crosses that ADB and disconnects on the next one. If the target signals Disconnect at Next ADB four or more data phases before an ADB (clocks 11 through 22 in Figure 2-74), the transaction disconnects on the ADB.

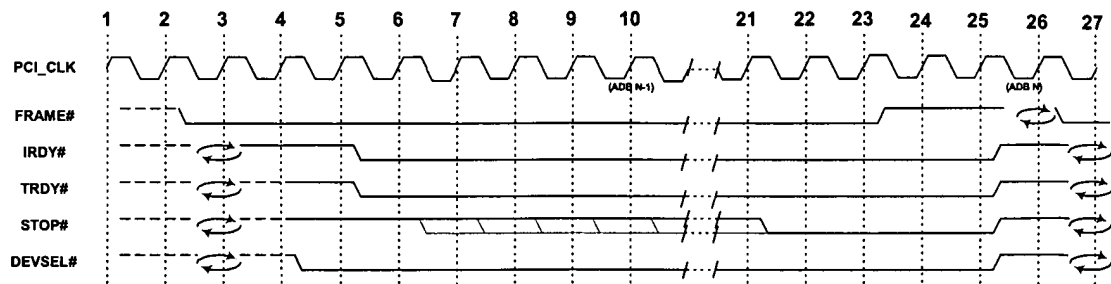


Figure 2-74: Disconnect at Next ADB on ADB N

Figure 2-75 through Figure 2-77 illustrate the target signaling Disconnect at Next ADB for transactions whose starting address is three, two, and one data phases from the ADB, respectively. (Note that the three- and one-data phase cases are not possible for 16-bit common-clock transactions because 16-bit common-clock transactions always transfer data in an even number of data phases.) In these figures, the target signals Disconnect at Next ADB on the first data phase. The initiator responds by deasserting FRAME# two clocks after the *first* data phase. The initiator deasserts IRDY# one clock after the *last* data phase but never less than two clocks after the *first* data phase (the clock in which FRAME# is deasserted).

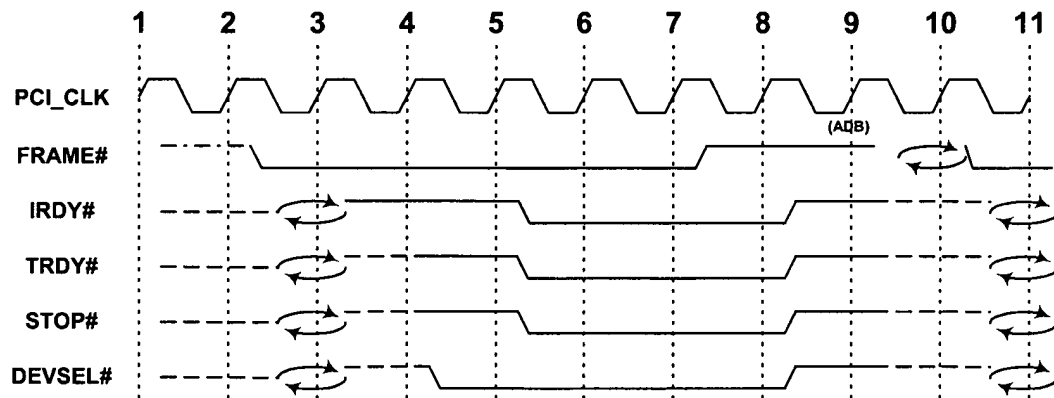


Figure 2-75: Disconnect at Next ADB with Starting Address Three Data Phases from an ADB

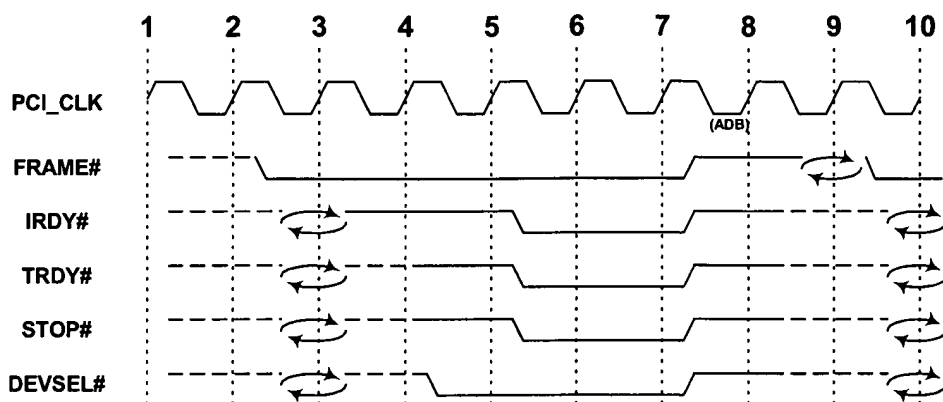


Figure 2-76: Disconnect at Next ADB with Starting Address Two Data Phases from an ADB

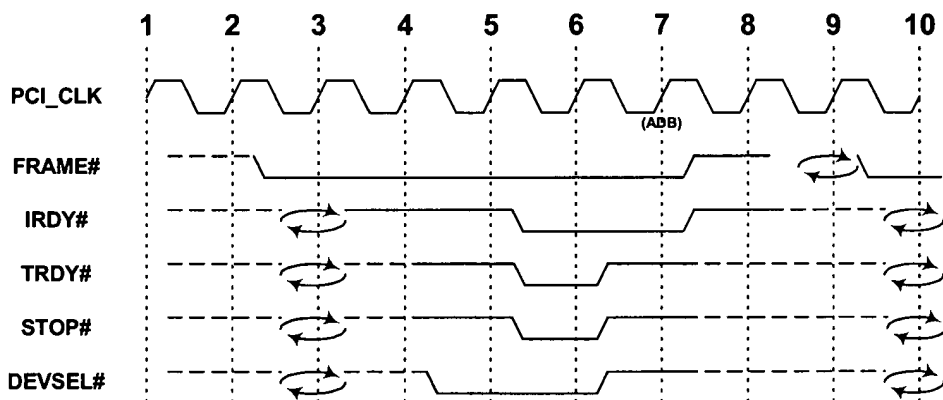


Figure 2-77: Disconnect at Next ADB with Starting Address One Data Phase from an ADB

### 2.11.2.3. Retry Termination

The target indicates that it is temporarily unable to complete the transaction by signaling Retry. The target signals Retry by asserting **STOP#** and **DEVSEL#** and keeping **TRDY#** deasserted on the first data phase of the transaction (with or without preceding wait states up to the maximum specified in Section 2.9.1). The target is permitted to terminate the transaction with Retry only under the following conditions:

- ☐ The device initialization time after the rising edge of **RST#** ( $T_{rhfa}$  specified in Table 2-7, “3.3V General Timing Parameters,” in PCI-X EM 2.0) has not elapsed.
- ☐ The device normally transfers data within the target initial latency limit listed in Table 2-12, but under some conditions that are guaranteed to resolve quickly, execution of the transaction would take longer. See Section 2.9.1 for additional limitations.

- ❑ The transaction is a memory write and all of the buffers for accepting memory write transactions are currently full with previous memory write transactions. See Section 2.13 for additional limitations.
- ❑ The transaction is not a memory write, it would require longer than the target initial latency to execute, and the target's Split Request queue is full.
- ❑ The transaction is a Split Completion, the target is a bridge as defined in Section 8.2, and the buffers for accepting Split Completions are currently full. See Section 8.4.5 for more details.

Unlike conventional PCI, a PCI-X target must not assume the initiator will repeat a transaction terminated with Retry. For transactions other than memory writes, the target must discard all state information related to a transaction for which it signals Retry. Delayed Transactions as defined in PCI 2.3 are not allowed. For memory write transactions, the target must not change its internal state in any way if it signals Retry on the first data phase of the Sequence. (The requester must deliver the full byte count of the Sequence after the first data phase is accepted. See Section 2.1.)

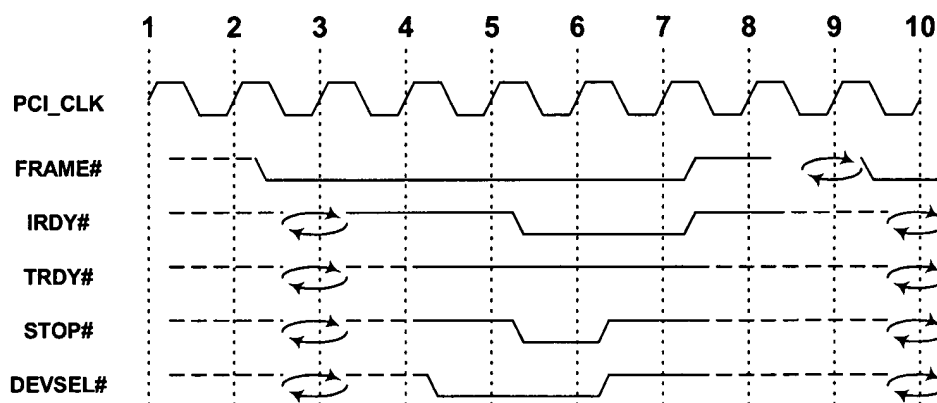


Figure 2-78: Retry Termination

#### 2.11.2.4. Split Response Termination

The target signals that it has enqueued the transaction as a Split Request by signaling Split Response. The target signals Split Response by asserting TRDY#, deasserting DEVSEL#, and keeping STOP# deasserted on the first data phase of the transaction (with or without preceding wait states up to the maximum specified in Section 2.9.1). (See Section 2.12.2.3.3 for the requirement for the target to signal Split Response for two clocks in 16-bit common-clock cases.) The target is permitted to signal Split Response on any DWORD transaction (except Special Cycle) and any read transaction. The target drives all bits of the AD bus high during the clock in which it signals Split Response for a read transaction.

Figure 2-79 shows Split Response for a read transaction (either burst or DWORD) in PCI-X Mode 1. Figure 2-80 shows Split Response for a DWORD write transaction in PCI-X Mode 1. Split Response in PCI-X Mode 2 would be identical, except the bus and control

signal driving and floating requirements are the same as for DWORD read and write transactions specified in Section 2.7.1.

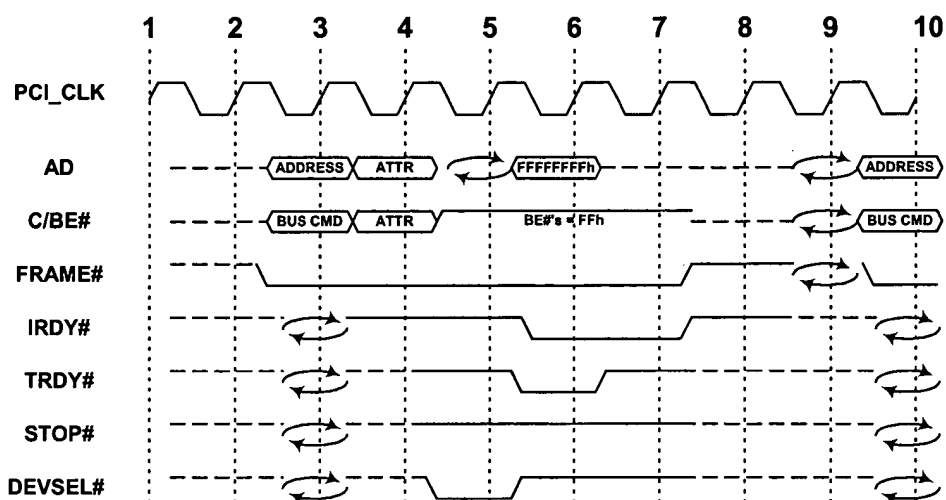


Figure 2-79: Split Response Termination for a Read Transaction, Mode 1

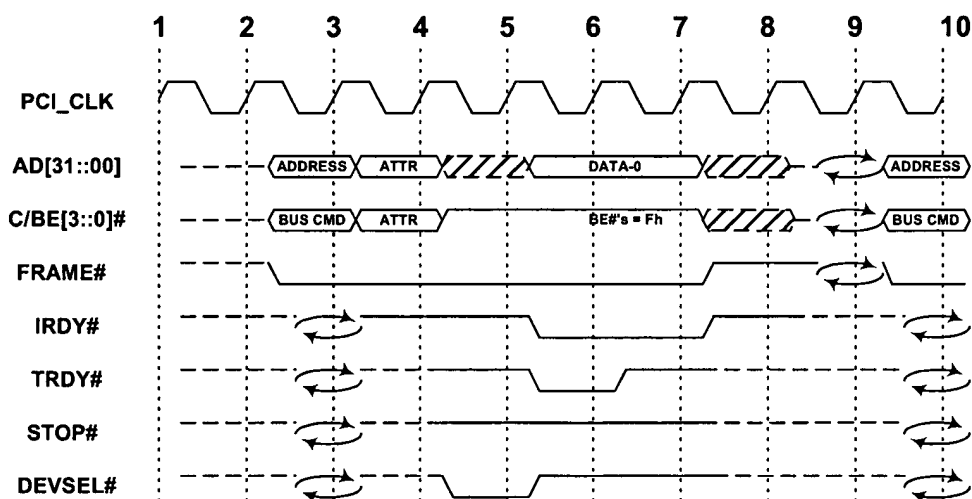


Figure 2-80: Split Response Termination for a DWORD Write Transaction, Mode 1

### 2.11.2.5. Target-Abort Termination

As in conventional PCI, the target signals Target-Abort to end the transaction and to notify the initiator not to repeat it. As in conventional PCI, PCI-X targets are permitted to limit the size and type of read transactions that they execute and to terminate all others with Target-Abort. For example, if a PCI-X device supports only DWORD read transactions in a certain address range, and if the device receives a read request for more than a DWORD, the

device is permitted to signal Target-Abort. See Section 2.10.1 for examples of the use of Target-Abort for read transactions that address both immediate-capable and split-only regions. (In some cases, independent memory write Sequences are combined by host or conventional PCI bridges, so targets are not permitted to use Target-Abort to limit the size of memory write transactions they execute.) The use of Target-Abort for Split Completion transactions is restricted. (See Section 2.10.5.)

It should be understood that signaling Target-Abort typically has deleterious effects on the system, possibly including halting execution of the system software, and device designers should avoid these circumstances whenever possible.

The target signals Target-Abort by asserting **STOP#** and deasserting **DEVSEL#** and **TRDY#**. The target is permitted to signal Target-Abort on any data phase. The transaction and the Sequence end on the clock in which the target signals Target-Abort regardless of its relationship to an ADB or the number of bytes remaining to be sent in the Sequence. The initiator deasserts **FRAME#** and **IRDY#** two clocks after the target signals Target-Abort, unless one or both of these signals deasserts sooner because the transaction was already about to end (e.g., byte count satisfied, initiator or target disconnection on an ADB).

Figure 2-81 illustrates a Target-Abort in the first data phase of a transaction. Figure 2-82 illustrates a Target-Abort after the target has signaled Data Transfer for several data phases of a burst transaction.

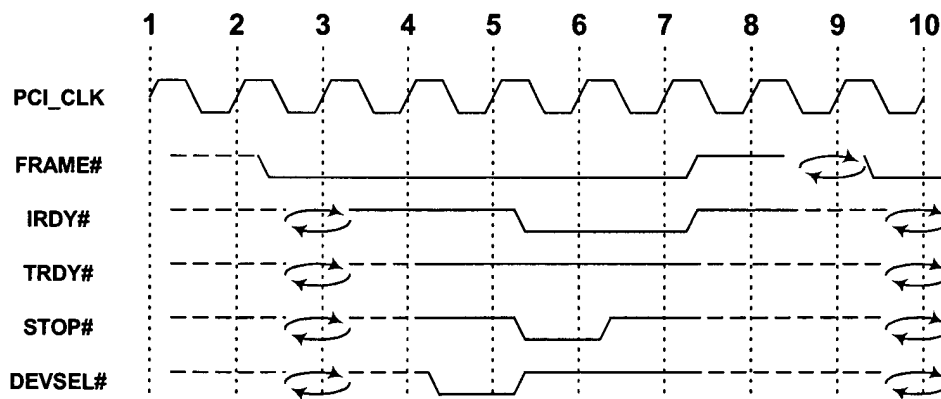


Figure 2-81: Target-Abort on First Data Phase